National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

NL-339 (r. 88/04) c

Canada

University of Alberta

Efficiency Aspects of Hypothetical Reasoning

by

Abdul Sattar

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Doctor of Philosophy

Department of Computing Science

Edmonton, Alberta
Spring 1992

National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-73055-2

Canada

THE UNIVERSITY OF ALBERTA

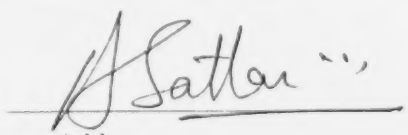
RELEASE FORM

NAME OF AUTHOR: Abdul Sattar
TITLE OF THESIS: Efficiency Aspects of Hypothetical Reasoning

DEGREE: Doctor of Philosophy
YEAR THIS DEGREE GRANTED: 1992

Permission is hereby granted to UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.



Permanent Address:
Village & Post: Khawas, Via: Kadera
Distt. Ajmer (Rajasthan) INDIA
Pin Code: 305 405

January 15, 1992

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **Efficiency Aspects of Hypothetical Reasoning** submitted by **Abdul Sattar** in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

R. Goebel

L. Yuan

J. You

M. Matthen

M. Ishizuka

Date: Dec 11/91

To my first teacher—my grand mother

Abstract

While representing and reasoning with prototypical information as hypotheses, the problem-solver has two main considerations: efficiently searching for a solution and reducing the total number of potential solutions (i.e., multiple extensions) which may be exponentially large. This dissertation presents a general problem-solving paradigm that is both representationally expressive and empirically efficient. The following are the main contributions of this research:

A Unified Framework: The proposed framework provides a new approach to truth maintenance systems based on consistency tree, which turns out to be a general paradigm to unify other useful empirical techniques such as meta-reasoning and constraint processing. Our approach overcomes many limitations of the existing systems as regards their expressive power and computational efficiency. Up to now, these techniques have been studied under separate research sub-area of AI. This research will help to resolve apparent differences and distinctions in these sub-areas.

Incremental Theory Selection: A rational approach based on crucial literals (cf. scientific experiments) is proposed to incrementally distinguish between competing theories, which, in turn, makes the reasoning more efficient. Its application in diagnostic reasoning is explored.

Acknowledgements

I am pleased to acknowledge my supervisor, Randy Goebel, for his never-ending intellectual, moral, ideological and financial support. He encouraged me to leave windy and humid Ontario for dry and cold Edmonton (which I enjoyed!). His insightful comments on my project for his AI course led to the development of the framework presented in this dissertation. I heartily thank him for all the suffering he endured to help me reach my potential; for the unrestricted access to all resources at his disposal; and for his unfailing confidence in my motivation and capability. Throughout my stay at the University of Alberta, he has served as my guardian with whom I fearlessly discussed all kinds of problems I faced in both personal and professional life. For all he has done for me, I cannot find enough words to express my gratitude.

I would like to thank my "Guru Brother" Scott Goodwin for listening to me whenever I needed it, for reading my drafts, and for his inspirations, straightforwardness, and friendship.

To Peter van Beek, Ron Kube, Robert Baron, Dekang Lin and Pavan Sikka, for reading my thesis drafts and for their valuable comments and constructive criticism.

To Anup Basu, for taking me out for frequent lunches at Angelo's Pizza.

To Mike MacGregor and Susan McGregor, for their great friendship.

To Tamer Ozsu, Keith Smillie, Jeff Pelletier, Jonathan Schaeffer, Ahmed Kamal, Lorna Stewart, Ehab El-Mallah, Steve Sutphen, Edith Drummond, Janet Service and Ollie Frost, for making me smile in normal day to day greetings.

Special thanks to my supervisory committee members Jia You, Li Yuan and Mohan Matthen, for reading my thesis drafts and for their moral support throughout my Ph.D. program.

To my many friends who shared good and bad times with me: Prahlad Rai, Brij Kishore, Hans Raj, George Ferguson, Andrew Edgar, Ramesh Sankaranarayana, Suryanil Ghosh, Charles Truscott, Aditya Ghose, Pat Fitzsimmons, Bonita Wong, Ajit Singh, Gurminder Singh, Manoj Jain, Narendra Ravi, Ashraf El-Nagar, Nilotpal Chakravarty, Daniel Lanovaz, Rasit Eskicioglu and Ken Schmidt.

A very special thank to Devendra Kaushik for his affectionate guardianship. Over the years, he has been my mentor and my friend. Also, I would like to mention G. V. Singh for his moral support and friendship.

My warm thanks to my parents for their moral and emotional support. This thesis would not have been possible without the support of my wife Nasreen and two children Kamal and Sarah. Nasreen has been a great source of inspiration throughout my stay in Canada. Her generosity and understanding is deeply appreciated.

Finally, I would like to thank the Canadian Commonwealth Scholarship Plan, the Department of Computing Science, the Faculty of Graduate Studies and Research, and (through my supervisor) the Natural Sciences and Engineering Research Council of Canada, and the Network of Centers of Excellence Institute for Robotics and Intelligent Systems for their financial assistance.

Contents

1 Introduction **1**

1.1 Artificial Intelligence and Logic 1
1.2 Reasoning with Incomplete Knowledge 3
1.3 What is Hypothetical Reasoning? 4
1.4 Expressiveness vs Efficiency 5
1.5 Motivation 8
1.6 Goals 9
1.7 Outline of the Presentation 10

2 The Computational Efficiency of Logic-based Reasoning **11**

2.1 Introduction 11
2.2 The Decision Problem 12
2.3 Asymptotic Complexity 14
2.3.1 Expressiveness vs. Tractability 15
2.3.2 Relaxing Correctness and Completeness 17
2.3.3 Decreasing Deductive Power 17
2.4 Empirical Efficiency 18
2.4.1 Meta-reasoning 19
2.4.2 Constraint Satisfaction 21
2.4.3 Truth Maintenance Systems 23
2.5 Conclusion 24

3 THEORIST: A Hypothetical Reasoning Framework **25**

3.1 Introduction 25
3.2 Scientific Theory Formation 26
3.3 The Basic Framework 28
3.4 Extended-THEORIST 31

3.5 The Efficiency Issues in THEORIST 35
3.6 Conclusion 36

4 Extended-Theorist: A Truth Maintenance System 38
4.1 Introduction 38
4.2 Background 40
4.2.1 Justification-based TMS 41
4.2.2 Logic-based TMS 44
4.2.3 Assumption-based TMS 45
4.3 Hypothetical Reasoning-based TMS 48
4.3.1 The basic architecture 49
4.3.2 Logical Specification 50
4.3.3 Truth Maintenance Tables 52
4.3.4 Incremental Explanation Construction 59
4.3.5 Updating the Truth Maintenance Tables 64
4.3.6 Empirical Efficiency 67
4.4 Relationship with Other TMSs 68
4.5 Summary 70

5 Constraint Satisfaction in Hypothetical Reasoning 71
5.1 Introduction 71
5.2 Constraints in Hypothetical Reasoning 75
5.2.1 Representing the CSP 75
5.2.2 Incremental Consistency Check 76
5.2.3 Forward Checking + Truth Maintenance 78
5.2.4 Unextendable Partial Explanations 80
5.2.5 Selecting Better Solutions 82
5.2.6 An Incremental Search Procedure 82
5.3 Network Consistency 88
5.3.1 Node-consistency 89
5.3.2 Arc-consistency 91
5.3.3 Path-consistency 92
5.3.4 k-consistency 92
5.3.5 What's a solution for a CSP? 93
5.4 ATMS-based Approach to CSPs 94
5.5 Constraint Logic Programming 96
5.6 Conclusion 98

6 Meta-reasoning: An Incremental Compilation Approach 99
6.1 Introduction 99
6.2 THEORIST: A Meta-reasoner 100
6.3 Incremental Compilation 102
6.3.1 Translating Meta-level Specification 104
6.3.2 Computing Inconsistencies and Potential Crucial Literals 104
6.4 Updating the Compiled Knowledge 107
6.5 Relationship with Prime Implicants 111
6.6 Conclusion 114

7 Using Crucial Literals to Select Better Theories 115
7.1 Introduction 115
7.2 Crucial Literals as Experiments 117
7.3 Identifying Crucial Literals 122
7.4 Diagnosing Multiple Faults 126
7.4.1 Medical Diagnosis 126
7.4.2 Full Adder Problem 129
7.4.3 The Multiplier Problem 129
7.5 Conclusions and Future Work 133

8 Conclusion 136
8.1 Summary of Contributions 136
8.2 Future Research 137

List of Figures

1 The complexity classes for decision problems 13

2 THEORIST 29
3 A Typical Interaction with Theorist 31
4 Consistency tree $tree(\neg t_1)$ 35

5 Problem-solver = Inference Engine + TMS 40
6 Environment Lattice 46
7 Comparing the JTMS and the ATMS nodes 47
8 Explanation construction in Theorist 49
9 Explanation generation in Extended-Theorist 50
10 An explanation tree and derived literals 53
11 Extending failure tree to consistency tree 56
12 Identifying potential crucial literals, insets and outsets 57
13 Incremental explanation construction for goal g 62

14 Constraints solving in hypothetical reasoning 75
15 Extending failure tree to consistency tree 79
16 Incremental Constraint Solving 85
17 Consistency trees in Four-Queens problem 86
18 State Transitions in Four-Queens problem 87

19 Theorist Database 102
20 The extra knowledge accumulated incrementally 103
21 Extending failure tree to consistency tree 106
22 Identifying nogoods caused by $\neg p1(a)$ 109
23 Consistency Trees 111

24 A circuit diagnosis problem 118

25 Circuit Diagnosis and Incremental Theory Selection 121
26 Consistency trees for t_1 and t_2 122
27 Consistency tree $tree(\neg t_1)$ 124
28 Diagnoses generated by Theorist and Extended Theorist . . . 128
29 A full Adder 129
30 Explanations generated by Theorist and Extended Theorist . 131
31 A Multiplier 133
32 Explanations generated by Theorist and Extended Theorist . 134

33 Revising Beliefs as Hypotheses 141

List of Examples

2.1 A first-order theory with finite domain 16
3.1 Tweety the non-flying bird 30
3.2 Theorist Database 34
5.1 The Four-Queens Problem 77
5.2 A simple CSP 78
5.3 A constraint network 90
6.1 Computing sum of two integers 101
6.2 Translating meta-sentences into object-level code 104
6.3 Compiling Theorist Database 105
6.4 Birds Fly 110
7.1 THEORIST Database for Circuit Diagnosis 119
7.2 THEORIST Database 123
7.3 Medical Database 127
7.4 Theorist Database for a Full Adder 130
7.5 Theorist Database for the Multiplier Problem 132

Chapter 1

Introduction

1.1 Artificial Intelligence and Logic

Artificial Intelligence (AI) as science is the study of computational intelligence, and as engineering is the study of techniques for building programs that solve intellectually demanding tasks [56]. Its central goal is to have a comprehensive theory of intelligence that can be used to make computers capable of intelligent behavior. The ability to behave intelligently depends on adequately representing knowledge about the world and then efficiently computing conclusions from that knowledge. In other words, AI is mainly concerned with representing facts and rules about the world in such a way that an intelligent machine can derive implicit knowledge about its domain by manipulating facts using rules of inference [111].

To represent our knowledge about the world, we need a language to communicate with computers. What kind of language should we have or can we have? Obviously, a natural language would be ideal. Unfortunately, today's computers are not capable of understanding such a powerful language. Thus we need a formal language defined over a set of symbols that can be understood by computers.

There has been widespread disagreement within the AI community about how knowledge should be represented (see [10, 11, 95]). However, there seems to be a general consensus that knowledge and beliefs should be represented

declaratively, and independently of how that representation is used to determine its intelligence—that is what Brian Smith calls the *knowledge representation hypothesis*:

> "Any mechanically embodied intelligent process will be comprised of structural ingredients that a) we as external observers naturally take to represent a propositional account of the knowledge that overall process exhibits, and b) independent of such external semantical attribution, play a formal but causal and essential role in engineering the behavior that manifests that knowledge." [169, page 2]

So, we need a well-specified formal language in order to show how expressions and symbols are related to individuals, relationships, actions and configurations in the world, i.e., a well-defined semantics is an essential part of a knowledge representation system [71, 72, 118, 124, 47]. This language should be expressive enough to be able to represent a wide-variety of knowledge about the real world. We also need an inference engine which can compute conclusions from the explicit knowledge base. Further, these conclusions should be made in a reasonable time, i.e., an intelligent agent must give timely responses.

The successful use of logic in formalizing mathematical reasoning[1] inspired AI researchers to consider logic as the fundamental tool to formalize intelligence. McCarthy's "Advice Taker" [111] seems to be the first to consider logic as the basis for a complete AI system. It consists of a general set of facts, represented as premises, and inference rules to manipulate them to derive conclusions [114, 116]. QA3 [65] was the first actual logic-based knowledge representation system in which knowledge was represented using the sentences of first-order logic and inferences were drawn by a complete first-order theorem prover. The fundamental significance of logic in AI is that it provides a clear, precise, elegant and intuitive way of assigning mean-

[1] Logic has been used as an appropriate tool to formalize the mathematical reasoning much earlier than the invention of computers and artificial intelligence. The use of formal logic can be traced back to the ancient philosopher Aristotle. It has been used in mathematical reasoning by many mathematicians and philosophers such as Leibniz, Frege, Russell, Peano and others. Also, formal logic was used by linguists to provide semantics for natural language understanding.

ing to symbols and of determining the validity of conclusions based on that meaning.

The earliest criticism of logic was that the combinatorial explosion of the search space generated by logical deduction is extremely large, i.e., theorem proving is too slow to be used for problem solving [121]. A more serious objection was the inability of classical logic to capture the forms of rational inference typically involved in commonsense reasoning (e.g., [120]). Logical reasoning is monotonic, sound and complete whereas human reasoning satisfies none of these properties. Some critics of logic view knowledge representation as entities to be manipulated by the programs—with no commitment to the meaning of symbols (e.g., [120, 66]). No matter how serious these criticisms are, the semantics for a knowledge representation language cannot be compromised because the lack of semantics is the fundamental source of confusion and misunderstanding [71] (see [95] for a nice debate over McDermott's critique of logic). Thus, our starting point should be a formal language which is at least as powerful as the first-order logic with equality [125]. Then the fundamental issues to be addressed are: the inability of classical logic to model reasoning in the absence of complete information (sometimes referred to as commonsense or human or nonmonotonic reasoning); and the computational cost of logical deduction.

1.2 Reasoning with Incomplete Knowledge

Like humans, an autonomous agent cannot be omniscient. Intelligent behavior by such an agent is based on its ability to reason in the absence of complete knowledge about its environment. To derive conclusions, a rational agent has to consider both supporting evidence and the absence of contradictory knowledge. The classical example is the flying ability of birds. If we know tweety is a bird, our commonsense conclusion is that tweety flies, since birds normally fly. Later, if we learn that tweety is a penguin, we must give up our previously held conclusion or belief.

To develop a formal understanding of such commonsense reasoning, several logic-based formalisms have been proposed for augmenting classical logic [145, 113, 119, 125, 33, 136, 134] such that tentative conclusions can be derived. These formalisms are based on two approaches for solving the problem of monotonicity [134].

The first approach is based on the argument that something is wrong with classical logic, so we need extra inference rules to handle nonmonotonic reasoning (e.g., *default logic* [145], *nonmonotonic logic* [119], *truth maintenance systems* [37], *autoepistemic logic* [125] and *conditional logic* [32]). The basic idea is to use the prototypical information of a domain of reasoning to fill in gaps in the knowledge base.

The second approach suggests that nothing is wrong with classical logic; but reasoning should not be viewed just as deduction from a knowledge base. For example, Circumscription [115] requires defaults as explicit assumptions (which can be expressed by adding the circumscription formula) to minimize the possible models. The basic intuition is that what is not known to be true must be false (cf., PLANNER's "THNOT" [74, 75], the closed-world assumption in deductive database [144], predicate completion using negation as failure [16] and circumscription [113, 115]).

Inspired by Popper's philosophy of scientific theory formation [142], and in the spirit of Israel [79], Poole et al. [136, 134] argue that rather than being a problem with logic, nonmonotonicity is a problem of how logic is used, and they proposed a hypothetical reasoning framework called Theorist.

In this dissertation, we consider the hypothetical reasoning based approach to nonmonotonic reasoning and show that hypothetical reasoning provides a general, simple, intuitive and empirically efficient way to solve the problems of Artificial Intelligence, particularly the problems of nonmonotonicity. The next section describes what we mean by hypothetical reasoning.

1.3 What is Hypothetical Reasoning?

Our knowledge has always been incomplete and is likely to remain so. Yet human intelligence has not given up real life activities; rather humans continue to act and make decisions despite a lack of complete information. To draw conclusions using incomplete knowledge, an intelligent agent has to use some kind of "guesswork" or "assumptions" based on prototypical or default information. These conclusions are always considered to be provisional and they may turn out to be incorrect.

> "At no stage are we able to prove that what we now 'know' is true, and it is always possible that it will turn out to be false. Indeed, it is an elementary fact about the intellectual history of mankind

> that most of what has been 'known' at one time or another has eventually turn out to be not the case." [103, page 26].

While questioning the appropriateness of the formalisms for nonmonotonic reasoning, Israel [79] has correctly pointed out that what is necessary for a formalization of our prototypical knowledge and the reasoning based on this knowledge is the specification of a set of rational epistemic policies. The main argument is that logical deduction is not a sufficient condition for rational belief; it is only good (sufficient and necessary) for making derivations in some formal system. In contrast, rational inferences are based on epistemic policies guiding belief fixation and that is what we often call the 'scientific procedure.' Further, he suggests that if real progress is to be made, nonmonotonic reasoning should be viewed as the problem of philosophy of science and epistemology, rather than as an extension of classical logic. In other words, scientific or commonsense reasoning, which can not be dealt with purely from a proof theoretical approach, must necessarily be heuristic in nature; that is not just the best we could hope for, but it is the only thing that makes sense.

In the same spirit, *hypothetical reasoning* stands for representing and reasoning with prototypical information as *hypotheses*. Hypotheses, when successful, could be used to explain the past and to predict the future [143]. The major tasks of hypothetical reasoning are: specifying the knowledge which is known to be true; the knowledge which can be assumed to be true; computing conclusions (i.e., explanations and predictions); of course, computing these conclusions efficiently; and discriminating between competing conclusions (i.e., selecting the best theory or revising imperfect theories). This form of reasoning is sometimes referred to as hypothetico-deductive reasoning, which is essentially a combination of deduction and abduction.

1.4 Expressiveness vs Efficiency

To represent a wide-variety of facts about the real world, an intelligent system must support a formal language that has an adequate expressive power. The fundamental question, often being asked, is what kind of features should a formal knowledge representation language have to satisfy the adequacy requirement. Responses to this question have created great controversy and confusion in the recent history of AI (e.g., [184, 66, 95]).

Nevertheless, there seems to be general agreement that in addition to the commitment to logical semantics—knowing what it means for a sentence to be true or false—the formal representation language must have at least all features of classical first-order logic with equality to describe incompletely known situations [124]. These features (conjunction, disjunction, quantifiers and negation) allow us to axiomatize the facts of some domain of discourse declaratively and separately, without worrying about how the processes manipulate that axiomatized knowledge.

Most critics of the above position argue that although logic is an appropriate tool to analyze knowledge representation and reasoning systems, the computational cost of logical deduction is intolerable (e.g., [127]). Some critics who (e.g., [117]) are completely disillusioned with logic claim that logic is unsuitable even for formalization of knowledge required by AI systems. While the former point is valid (i.e., non-controversial) and many attempts have been made to improve the efficiency of logical deduction (e.g., [70]), the latter criticism has no basis as shown by a flurry of responses to McDermott's critique of logic [95]. Therefore, a formal language based on first-order logic is the only answer to the question of expressive adequacy. The hypothetical reasoning system being considered in this thesis supports a full first-order logic based language, in which the facts, rules and hypotheses can be expressed using clausal sentences or well-formed formulas.

Unfortunately, the decision problem for first-order satisfiability and validity is semi-decidable—determining whether a given sentence is not a consequence of a set of axioms is not effectively computable. This has led to a fear over whether it is worthwhile to have representational adequacy of AI systems. Nonmonotonic formalisms can easily be shown to be intractable (i.e., propositional logic based frameworks are decidable but computationally intractable) to semi-decidable (i.e., circumscription based formalisms) to undecidable (i.e., consistency based formalisms). The question is: should we abandon a formal approach to nonmonotonic reasoning? Of course, our answer, along with that of many others, is "no." Because abandoning a logical approach to AI is not a solution to the problem, rather, it is abandoning the problem itself. Consequently, the real progress in the development of reasoning systems can only be made by working on both their representational adequacy and computational efficiency.

Undoubtly, one of the most serious concerns in AI today is transforming the formal work to practice by finding efficient computational methods.

There are two different perspectives on this issue. The first is traditional complexity theory, which provides a wide variety of tools to classify and predict the behavior of effectively computable procedures—algorithms. These tools are relatively well-developed, but totally inappropriate if the system of reasoning under consideration suffers from any aspect of undecidability. The second is to use techniques such as truth maintenance, constraint satisfaction and meta-reasoning to control and limit unrestricted reasoning with restricted (i.e., finite) resources. The problem is to control the combinatorial explosion of possibilities such that the search in an exponentially or infinitely growing space can be made relatively efficient.

Recent work on tractability suggests a number of approaches to ensure that reasoning systems compute within predictable bounds. For example, weakening the expressive power of the language (e.g., [93, 165, 94]); relaxing the correctness and completeness requirements (e.g., [86, 87]); and abandoning the deductive power of classical logic (e.g., [129]). Our main objection to these approaches is that by imposing such restrictions on the logical system, we are defeating the purpose of having formalisms in the first place. The believers in logic have been arguing hard to convince the sceptic that real progress cannot be achieved without having representational adequacy. Without weakening the power of the language, one use of traditional complexity measures may be in identifying algorithmic subcomponents of the reasoning system which can be improved by analyzing its asymptotic behavior [61], e.g., Schubert's special inference methods for reasoning about types, parts, color, and time [162].

Alternatively, we can think of the inherent intractability in a decidable or undecidable logical framework as the worst case scenario; and we can be more optimistic about the average cases. If the problem of computational efficiency is viewed as a search problem, then finding better control for deduction becomes the main concern. There have been much work for increasing the efficiency of deduction by using special cases of reasoning in logic and preserving the semantics (e.g., [85, 105, 171, 172]); and by specifying procedural control of deduction at the meta-level (e.g., PLANNER [74], GOLUX [70], AMORD [28] and MRS [15, 55]). Other efforts involve various intelligent search strategies incorporated in the logic programming systems (e.g., [174, 21, 107, 54]), truth maintenance systems (e.g., [37, 26]), and constraint satisfaction techniques (e.g., [101, 45, 67]).

1.5 Motivation

From the logicist view point, the task of knowledge representation is to declaratively formulate facts about the real world such that domain knowledge and control knowledge can be treated separately. The two distinct types of knowledge are sometimes referred to as epistemological and heuristic parts of the reasoning systems:

> "The epistemological part of AI studies what kinds of facts about the world are available to an observer with given opportunities to observe, how these facts can be represented in the memory of a computer, and what rules permit legitimate conclusions to be drawn from these facts. It leaves aside the heuristic problems of how to search spaces of possibilities and how to match patterns." [112]

The Theorist hypothetical reasoning system [136, 134] adopts a scientific theory formation approach to nonmonotonic reasoning that involves not only observing and collecting facts about the domain of discourse, but also hypothesizing, constructing explanations (or making predictions), and testing and selecting (or revising) theories. Theorist uses a first-order logic based language that seems to satisfy the basic representational adequacy requirement for the intelligent systems. However, there are some epistemological issues yet to be settled, such as specifying theory preference using domain dependent/independent knowledge; and the semantics of the preferred theories (see [62] for some efforts to address these issues). Like most nonmonotonic reasoning systems, efficiently computing conclusions (i.e., constructing explanations) is a major concern of the Theorist hypothetical reasoning system.

The motivation of this work is to attempt to address both epistemological and computational issues, e.g., understanding how some crucial knowledge about a particular domain can distinguish between two mutually incompatible theories. Our primary motivation is to seek an empirically efficient hypothetical reasoning system that allows us to experiment with potential specifications and various domain-dependent heuristics.

1.6 Goals

Since explanation in Theorist depends on logical satisfaction and consistency, the use of a logical language as powerful as first-order logic makes the system undecidable. Consequently, traditional complexity theory is void for the hypothetical reasoning system under consideration unless some aspects of the representation system are weakened to make it decidable. Our main concern for investigating the efficiency aspects of hypothetical reasoning systems is that one should not sacrifice representational adequacy of the reasoning system just for claiming tractability or decidability. Rather, our quest should be some pragmatic methods to gain computational efficiency while retaining the logical richness of the hypothetical reasoning system.

The basic computational advantage in the Theorist hypothetical reasoning system is due to its goal-directedness, i.e., irrelevant information need not be considered in the derivation tree. However, the consistency check for hypotheses is unavoidable and very expensive—computationally semidecidable. Our view is that the semidecidability is not a problem but an indication of the wealth of information potentially uncovered during consistency checking.

Precisely, our goals are summarized as follows:

- Extending the consistency check mechanism in Theorist to identify the crucial information that may be used to discriminate between competing theories.
- Identifying and recording inconsistent combinations of hypothesis instances as TMS-like nogoods, and reusing partial derivations to avoid unnecessary search.
- Representing and reasoning with a system of constraints within the hypothetical reasoning framework such that various heuristics for constraint solving can be integrated to improve search efficiency.
- Specifying control rules at the meta-level in hypothetical reasoning to improve the efficiency of deduction; and proposing an incremental compilation approach to meta-reasoning.
- Unifying theory preference/selection criteria and employing them to incrementally discriminate between incompatible theories.

- Implementing a prototype system in Quintus Prolog and then studying empirical performance of the hypothetical reasoning system by considering some real world problems such as medical or circuit diagnosis, constraint satisfaction in computer vision and planning.

1.7 Outline of the Presentation

Chapter 2 reviews the fundamentals of computability and how efficiency of logic-based reasoning systems can be addressed both from theoretical and empirical perspectives.

Chapter 3 describes the basic Theorist hypothetical reasoning framework and the philosophical basis behind it, along with an extension to the basic architecture. Then it describes how efficiency of the extended Theorist can be improved.

Chapter 4 first discusses the motivation for early truth maintenance systems (TMSs) and surveys the main families of TMSs such as justification-based TMS, logic-based TMS and assumption-based TMS. Then a logical framework for a hypothetical reasoning-based TMS is described and compared with the traditional TMSs.

Chapter 5 develops a hypothetico-deductive approach for adequately specifying and efficiently solving constraints. A correspondence between the notion of consistency in a network of relations [101] and consistency checking in the hypothetical reasoning system is established. It is shown that most of the constraint solving heuristics can be integrated along with truth maintenance to improve the efficiency of the constraint solver. Hypothetical reasoning turns out to be a general approach for constraint solving; i.e., suitable for single solution, multiple solutions or multiple constraint satisfaction problems over the same system of constraints.

Chapter 6 examines the role of the Theorist hypothetical reasoning system as a meta-reasoner and discusses an incremental compilation approach (in contrast to the purely interpretative or compilation approaches) along with an update method for compiled knowledge.

Chapter 7 presents a rational solution to the multiple extension problem based on crucial literals.

Finally, chapter 8 presents a summary of contributions of this research and discusses directions for future research.

Chapter 2

The Computational Efficiency of Logic-based Reasoning

2.1 Introduction

Although there is disillusionment amongst some, the general AI community seems to have warmly embraced classical logic with standard model-theoretic semantics as the fundamental tool for analysing and developing intelligent systems. The classical first-order logic (FOL) provides adequate expressive power, an elegant semantics and deductive capabilities. These are the most essential requirements for a formal language to be able to specify and reason with knowledge about some domain. The syntactical structure in FOL consists of predicate symbols, function symbols and variables. The terms and formulas can be constructed by using function applications, atomic formulas, quantifiers and logical connectives such as negation, disjunction and conjunction. The semantics involves the assignment of meaning to symbolic structures (referred to as Tarskian semantics), typically consisting of a domain, and functions and relations over that domain. Interpretations are mappings from a symbolic structure to 'true' or 'false'.

The major operation performed by the reasoning systems based on such a logical language is to determine whether a given formula follows from other formulas. In other words, determining whether a given formula is a theorem.

This chapter addresses the issues concerning efficiency: can we predict the behavior of a first-order logic based reasoning system? What are the worst and average cases? If the theoretical methods used to answer such questions are not sufficient, then what is the alternative for speeding up reasoning?

2.2 The Decision Problem

The problem of finding an effectively computable procedure for determining the truth or falsity of a whole class of statements is called a *decision problem*. For example, a knowledge representation system has to answer a fundamental question: "Is it possible to determine whether or not a given statement pertaining to a field of knowledge is a consequence of the axioms?" [76]. In mathematical logic, this is the problem of determining whether or not a given formula is *universally valid* (or *satisfiable*) with respect to a given set of logical sentences.[1] Both of these problems are equivalent: a formula Φ is not universally valid in a domain of individuals iff $\neg\Phi$ is satisfiable in that domain. Similarly, the universal validity of $\neg\Phi$ and the satisfiability of Φ are contradictory to each other. In logic, these two problems of universal validity and satisfiability are referred to as by a common name **the decision problem** [76].

In the 1930's Church concluded that the quest for a general solution of the decision problem in the predicate calculus is hopeless [76]. However, depending on the degree of solvability, the decision problems can be characterized as decidable, semi-decidable and undecidable. These terms can be defined using a characteristic function [22], $C_M(X)$, for an n-array predicate $M(x_1, ..., x_n)$ such that $X = (x_1, ..., x_n)$.

$$C_M(X) = \left\{ \begin{array}{ll} 1 & if\ M(X)\ holds \\ 0 & if\ M(X)\ doesn't\ hold \end{array} \right\}$$

If the characteristic function C_M is computable, then the predicate $M(x)$ is said to be *decidable*, i.e., it returns 1 when $M(X)$ holds and 0 when it doesn't hold. If it is not decidable, then it is *undecidable*. The predicate

[1] A formula is universally valid if it is deducible from the axiom system (i.e., facts and rules of inferences) in every domain of individuals. A formula is satisfiable if there exists a domain of individuals in which it is satisfiable.

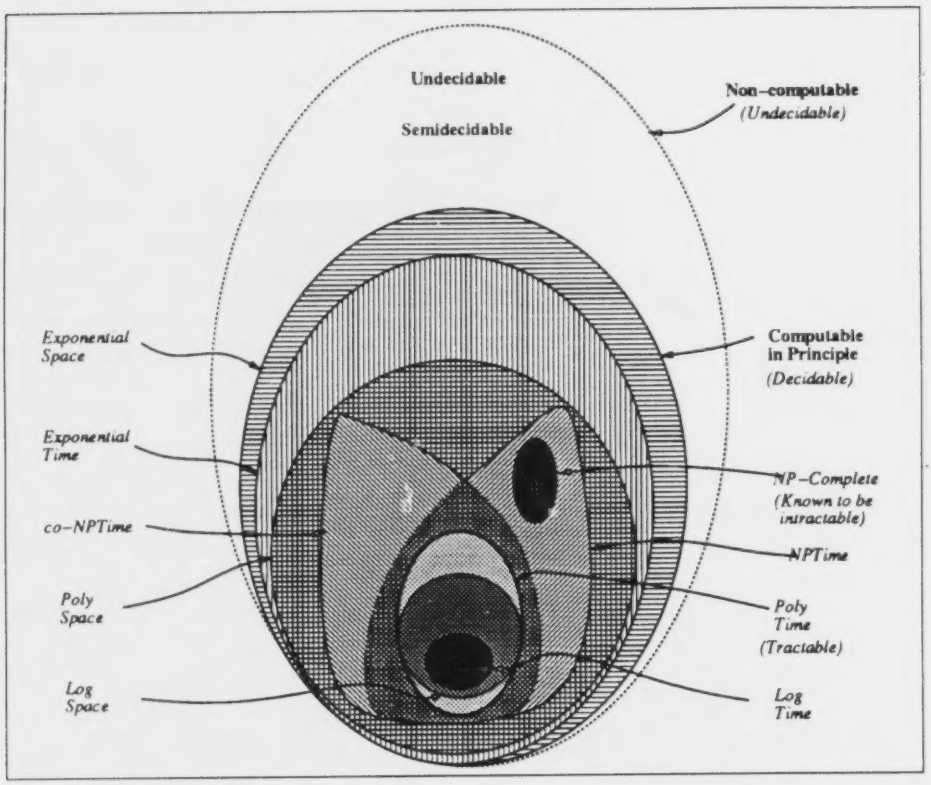


Figure 1: The complexity classes for decision problems

$M(x)$ is *semi-decidable* if its characteristic function C_M is computable only if $M(x)$ holds, i.e., it returns 1 if $M(X)$ holds and it is undefined otherwise.

Finding the most efficient method for solving a decision problem has been a main concern of computer scientists. However, it is not always possible to find an efficient algorithm, or to prove whether we can define an effectively computable procedure to solve a given problem. There are problems which are very very hard to solve—*intractable*.

The earliest intractability results (e.g., [177]) show that it is impossible to find an effectively computable procedure for undecidable problems. From a computational efficiency viewpoint, the difference between the semi-decidable and undecidable problems seems to be moot. It is possible to define an effectively computable procedure for decidable problems; but it does not guarantee tractability of such procedures. Tractability usually identified with polynomial time algorithms. Some of the decidable problems cannot be solved in polynomial time using a non-deterministic computing machine; this has led to the theory of NP-completeness [19, 53].

Figure 1 shows the complexity classes of decision problems [53, 68]. Only the class of problems that are solvable in polynomial time makes sense in practice—sometimes referred to as a class of tractable problems. The problems from the rest of the complexity classes are practically very hard or possibly intractable whether they are decidable or undecidable. However, the average case may not be as bad as the worst case analysis [93].

There seems to be only two ways to address the problem of efficiency of the first-order logic based reasoning system: first, restricting the language such that asymptotic complexity theory can be applied; and second, applying heuristic-based techniques to improve empirical efficiency without weakening the language.

2.3 Asymptotic Complexity

Asymptotic complexity theory is concerned with the analysis and synthesis of algorithms with respect to their resource requirements. Insights from that analysis often leads to more efficient algorithms. However, these measures are totally inappropriate if the formal language used for the reasoning system has any aspect of undecidability.

To analytically predict the computational behaviour, the first-order logic

based knowledge representation systems may be weakened in a number of ways, such as the way we describe the knowledge, and the way we want to make inferences. Recent analysis concerning tractability (or decidability) of the logic-based reasoning systems are broadly based on [94, 40] limiting expressiveness; or relaxing correctness and completeness of the inference engine. The first argues that the computational efficiency and representational adequacy of a reasoning system are mutually incompatible. Thus, there exists a tradeoff between expressiveness and tractability [93, 94, 165]. The second suggests that the notion of correctness and completeness should be relaxed (e.g., [91, 87]). In addition, some attempts have been made to use deductively weak logic such as relevance logic [4] for knowledge representation (e.g. [128]) to achieve computability.

2.3.1 Expressiveness vs. Tractability

The idea of tradeoff between expressiveness of the logical system and its computational tractability is not a new one. This issue has been addressed quite well in the literature of philosophy (e.g., Russell's Theory of Description [82]) and mathematical logic (e.g., first-order theories with finite domain [76, 1]).

For example, consider a case in which first-order language is restricted such that the domain of individuals contains a finite number of members, then the solution for the decision problem can be found. The idea is that a formula of FOL can be reduced to a formula of sentential calculus by replacing the universal quantifiers by a finite conjunction and the existential quantifiers by a finite disjunction (See Example 2.3.1).

A subset of first-order logic, namely Horn clause logic, has been the basis of logic programming languages such as Prolog systems [88]. Although the knowledge representation system based on Horn clauses is less expressive than the first-order logic-based language, it is possible to provide a natural procedural interpretation which corresponds to SLD-resolution (SL-resolution for definite clauses) [98]. This is, in turn, based on Robinson's resolution principle [151]. In general, reasoning with Horn clause theories is not only intractable, but also non-computable [176]. However, further restrictions may lead to tractability of testing the satisfiability (e.g., [35]).

We can always find some restricted classes of FOL for a finite domain of individuals for which the decision problem is solvable (e.g., solvable cases

Let us consider a formula: $\exists x \forall y (F(x,y) \wedge \neg F(x,y))$.
Domain of individuals is { 0,1 }.
First, we replace existential quantifiers by disjunction,
$\forall y (F(0,y) \wedge \neg F(0,y)) \vee \forall y (F(1,y) \wedge \neg F(1,y))$.
And then we replace universal quantifiers by conjunction
$(F(0,0) \wedge \neg F(0,0) \wedge F(0,1) \wedge \neg F(0,1)) \vee (F(1,0) \wedge \neg F(1,0) \wedge F(1,1) \wedge \neg F(1,1))$.
Hence, the satisfiability of a formula in a domain of two individuals is reduced to the satisfiability of the sentential formula of the form
$(A \wedge \neg A \wedge A \wedge B) \vee (C \wedge \neg D \wedge C \wedge \neg C)$.

Example 2.1: A first-order theory with finite domain

of undecidable problems [1]). However, not every decidable problem is computationally tractable. The satisfiability problem in propositional logic is computationally intractable—NP-complete. Recent studies on tractability of default reasoning using propositional Horn clause theories indicate that the problem of determining a most-preferred model is intractable (e.g., [165]). Even if we restrict the propositional Horn theories to allow only normal unary defaults, the problem still remains intractable [173]. Further restrictions like imposing acyclicness on Horn clauses make inference computation tractable [165]. Also, an effective procedure has been proposed to generate all consistent extensions for finite theories (i.e., theories with only finite numbers of variables, constants, predicates, defaults and no function symbols) in default logic [39].

In other efforts, emphasis has been given to express knowledge about the world only in terms of *explicit* beliefs such that the logical entailment is reduced to *database retrieval* (e.g., 'vivifying' (filling out) an incomplete knowledge base) [91, 92, 94, 40]). The idea is that "The expressive power of FOL determines not so much what can be said, but what can be left unsaid" [93]. Thus, if we leave nothing unsaid in the knowledge base, then the reasoning system should be able to effectively compute conclusions. This seems to have a close relationship to much earlier work on "diretedness" of

expression (cf. "What is Russell's Theory of Description?" [82]).

2.3.2 Relaxing Correctness and Completeness

One of the fundamental requirements for a logic-based reasoning system is a sound and complete inference engine. Since behavior of such an inference engine cannot be predicted in general, attempts have been made to relax soundness (e.g., nonmonotonic inference rules such as negation as failure [16] and default rules [145]) and completeness (e.g., depth-first search or considering only explicit beliefs [91]) to somehow control the inference mechanism to be able to perform in the desired way. The reasoning process may be restricted to terminate arbitrarily after a fixed amount of time or inferences. The conclusions derived under these limitations are correct, but not all of the correct conclusions can be computed.

Under one such theory, the complexity of reasoning, when only explicitly represented beliefs are considered in contrast to the implicit beliefs, is given by the following theorem:

Theorem 1 ([91]) *Suppose KB and α are propositional sentences in conjunctive normal form. Determining if $KB \models \alpha$ is co-NP-complete but determining if $KB \Rightarrow \alpha$ has an $O(mn)$ algorithm, where $m =\mid KB \mid$ and $n =\mid \alpha \mid$.* [2]

While using full expressive power of the first-order logic, an attempt to restrict the deduction steps—relaxing the completeness has been made in [87]. For some, a formal model of belief, called a *deduction model*, for representing situations in which belief derivation is logically incomplete is described and it claimed that the deduction rules can be complicated, but they are effectively computable functions.

2.3.3 Decreasing Deductive Power

In mathematical logic, the use of material implication has been under criticism since the time of Aristotle because it does not model the intuitive notion of implication. Anderson and Belnap [4] present relevance logic, which

[2] $\Rightarrow$ is a provably tractable subset of $\models$. Thus, conclusions are sound, but not logically complete.

is claimed to capture the intuitive notion of implication by weakening the standard implication. In relevance logic, *modus ponen* is not a valid rule of inference, i.e., the deductive power of the classical logic is lost. For example, given Tweety is a bird and that all birds are animals, $bird(Tweety)$ and $\forall x \neg bird(tweety) \vee animal(x)$, relevance logic does not support the conclusion that $Tweety$ is an animal.

The effect of relevance logic on tractability is shown by developing propositional tautological entailment [91]. The decision problem of whether a given sentence in conjunctive normal form follows from another is solvable (in polynomial time) in this logic, whereas it is likely to be intractable in standard propositional logic. Patel-Schneider [128] has developed a decidable logic—first-order tautological entailment. Its goal is to have the expressiveness of first-order logic and computational advantage of weakening standard implication.

Further, a *hybrid logic* is proposed to strengthen the inference mechanism of the above decidable variant of the first-order tautological entailment, while retaining the decidability [129]. This logic is created by combining two components called the assertional component and the terminological component. The assertional component is the same as the decidable form of first-order tautological entailment. The terminological component is concerned with the formation of concepts and roles, and definitions of one or two place predicates in terms of concepts and roles.

Logical consequences in hybrid logic follow from the definitions of the two components. If we reconsider the same example, then $animal(Tweety)$ follows from $bird(Tweety)$ under the definition $bird \leq animal$. If in all models $bird \leq animals$ is true, then whenever $bird(Tweety)$ is true $animal(Tweety)$ is true.

2.4 Empirical Efficiency

From the logicist viewpoint, the task of knowledge representation is to declaratively formulate facts about the real world such that domain knowledge and control knowledge can be considered separately. The domain knowledge specifies a conceptualization of a domain, whereas the control knowledge specifies rules to manipulate the domain knowledge. The two distinct types of knowledge are sometimes referred to as epistemological and heuristic parts of the

reasoning systems [112].

While the first-order logic seems to satisfy the epistemological requirement, computational tractability can only be achieved by weakening some aspects of FOL as discussed in the previous section. Alternatively, without weakening the epistemological component of a reasoning system, the computational efficiency can be addressed using heuristic-based techniques—that is what we call *empirical efficiency*.

Historically, the heuristic-based techniques have been extensively used to study the methods and rules of discovery and inventions in mathematics, logic, philosophy and psychology [130]. Here, our main goal is to find some clever ways to control and limit unrestricted reasoning with restricted (i.e., finite) resources. The efficiency problem can be cast in terms of the problem of efficiently searching in the exponentially growing search space. Most of the empirical techniques, proposed during the last two decades in AI, may be characterized under three headings: meta-reasoning, i.e., specifying meta-knowledge to control the deduction process [70, 74, 28]); constraint satisfaction, i.e., a system of constraints can be used to reduce the search space (e.g., [101]); and truth maintenance systems [37, 26]).

2.4.1 Meta-reasoning

A meta-level reasoning system is a system which performs reasoning about reasoning. In the conceptualization of meta-knowledge, expressions, subexpressions and sequences of expressions are treated as objects of the domain of discourse; functions and relations are defined over these objects [56, Chapter 10]. The use of meta-reasoning in AI was first suggested by McCarthy [111].

Most representation and reasoning systems may be viewed in terms of two levels of reasoning: *meta level* and *object level*. For example, in a blocks world, specifying the physical movement of a robot arm would be at object level, and the specification of all planning tasks (i.e., how to decide an action to be performed at object level) would be at meta-level [55]. However, there may be many levels of reasoning. The distinction between the meta-level and the object-level is arbitrary and relative.

The primary motivation for using meta-level reasoning as the basis of intelligent systems is to have a declarative language with high expressive power, which allows us to easily and quickly specify the meta-knowledge about the domain of discourse, and with a capability of explicitly and sepa-

rately representing control knowledge which enables the reasoning system to be computationally efficient. The necessity of meta-reasoning in the deductive systems is clearly and strongly advocated by Pat Hayes [72]:

> " A system which makes inferences to generate new facts must control its inference-making capabilities in some way. This control itself requires the storing and using, by the system, of information about the deductive process. That is: the system must represent and use knowledge about its own behavior." [72]

The main idea is to allow the user to explicitly represent deductive control. That is, to describe how the interpreter behaves while constructing proofs. When added to the reasoning system, control information constrains its behavior to produce solutions more efficiently.

The literature contains a number of meta-level reasoning systems that provide many ways of explicitly representing meta-knowledge and controlling the inference mechanism at the object level. Van Harmelen [178] presents a classification of meta-level architectures. The classification is based on the notion of what he calls *locus of action* which indicates the place of activity in the system. Here, activity means interpretation of the instructions at the meta-level or at the object-level or at both levels. Depending on this characteristic, van Harmelen has classified meta-reasoning systems into three classes: object-level inference systems (e.g., GOLUX [70], the Prolog system [50, 51]), mixed-level inference systems (e.g., TEIRESIAS [24], NEOMYCIN [15] and MLA [55]) and meta-level inference systems (e.g., PRESS [170], the amalgamated Prolog system by Bowen and Kowalski [9], and Socrates [20]).

The logic programming systems (i.e., Prolog) consider a subset of first-order logic, Horn clauses, and have only a depth-first, left-to-right inference strategy. Therefore, despite their deductive capability, these systems are quite inefficient as regards to their computability and flexibility of control. The use of meta-programming has been shown to be quite promising for solving the problems of the Prolog-like systems (see [99]).[3]

A major problem with meta-programming techniques is the inherent lack of efficiency, which arises due to a substantial overhead in handling object-level variables while the meta-level interpreter executes the object program.

[3]A regular biennial workshop is being held on meta-programming that covers many aspects of meta-reasoning.

This problem can be avoided if we compile the meta-level specifications into the object program, and then execute the compiled code directly and efficiently on current Prolog systems.

However, both interpretive and compilation approaches are complementary with respect to their advantages and disadvantages [175]. The interpretive approach is less efficient, but allows user to understand the behavior of the inference mechanism which can be used to modify the control. On the other hand, compilation is efficient, but it is difficult to understand the compiler and the compiled code. The partial evaluation technique suggested in [175] is an attempt to take advantages of both of these approaches without sacrificing the efficiency. In the same spirit, chapter 6 of this dissertation proposes an incremental compilation approach that improves efficiency of meta-reasoning especially in the nonmonotonic reasoning environment.

2.4.2 Constraint Satisfaction

The word constraint, is a general term, and may be defined as a relation over a Cartesian product of sets, a Boolean predicate, a fuzzy relation, an algebraic equation, an inequality, a Horn clause in a logic program and various other arbitrary complex symbolic relationships [102]. Essentially, a constraint specifies a kind of meta-knowledge. Constraint-based reasoning may be viewed as a form of meta-reasoning. However, it has mostly been studied as a separate sub-area of AI.

The notion of constraint satisfaction in AI suggests that we represent our knowledge about the world in terms of what is allowed and what is not allowed [102]. When we formulate the reasoning tasks in terms of a system of constraints, specialized techniques for efficiently solving the resulting constraint satisfaction problems (CSPs) can be directly applied to AI problems such as computer vision [23, 14]. There have been two most commonly used CSP techniques, *backtracking* and *consistency checking*.

Generally, backtracking sequentially instantiates the variables from ordered representations of their domains. When all of the variables of a constraint relation are instantiated, then its truth value is tested. If it is true, then the process of instantiation and testing will continue. Otherwise, the process fails back to the last variable instantiated that still has untried values in its domain, and re-instantiates the variable to its next value. The automatic backtrack search mechanism of the Prolog systems naturally supports

this approach.

The backtracking strategy is the most efficient from a space point of view. But, it suffers from well-known *thrashing* behavior. The time taken to find a solution tends to be exponential both in the worst case and on average [101, 102, 67, 45].

Apparently, the first attempt to overcome the inefficiencies of backtracking was made in [183, 123, 101]. The family of algorithms based on these ideas are called consistency techniques. For example, constraint propagation is one of the consistency techniques used to narrow down the search space in an *a priori* manner by removing inconsistencies [45]. To do so, a CSP is represented as a network in which each node represents a variable and a domain of values associated with it. Basically, these techniques remove local inconsistencies, which may cause inconsistencies in the neighboring arcs, which are, in turn, removed such that eventaully a consistent instance of the network is obtained. Mackworth [101] presents network consistency in terms of node consistency, arc consistency, and path consistency. Further improvements and use of a number of heuristics are described in the literature [45, 67, 102]

Logic programming systems (e.g., Prolog) are considered to be most suitable candidate for representing the CSPs [49, 90, 181]. But the automatic backtracking control mechanism of standard Prolog makes it computationally inefficient. Standard Prolog cannot actively use constraints to eliminate inconsistent combinations of values for variables. Rather, it generates (instantiates all variables) values for the predicates and then tests the truth value. Some improvements over the conventional backtracking strategy have been proposed: Intelligent Backtracking, Selective Backtracking, Dependency Directed Backtracking and backmarking [21, 107, 54]. These improvements do not attack the real problem of the backtracking mechanism. They try to reduce the search space in an *a posteriori* way, i.e., after having discovered a failure. In addition, the reasons for dead-ends may be recorded to avoid rediscovery of the same backtrack points [31].

Dincbas et al. [180, 34] proposed some constraint extensions to standard Prolog. The main objectives of these extensions are to retain the expressive power of Prolog and to efficiently solve CSPs. The idea is based on the consistency techniques (the second approach to the CSPs). When we integrate the consistency techniques into the Prolog control mechanism, then it enables Prolog to reduce the search space in an *a priori* way, i.e., before discovering a failure. This is accomplished by removing inconsistent combinations of

values of the variables in a predicate (a constraint relation). This leads to an early detection of failures and reduces the amount of backtracking as well as the number of consistency checks.

2.4.3 Truth Maintenance Systems

The development of truth maintenance systems (TMSs) has been an active research area since the late seventies ([37, 26, 44, 106]). The idea is that the problem-solving system must be viewed as a combination of two components: 1) the inference engine, which manipulates the knowledge about a specific problem domain using some rules of inference, and 2) the truth maintenance system, which records dependencies among propositions and maintains a consistent view[4] of the knowledge base. The underlying intuition for most TMSs is that, while searching for a solution, the problem-solver encounters many states which are distinct but very similar. The obvious question is how to make use of these similarities such that the same state is not revisited and the mistake (if any) once made is not repeated.

The fundamental goals of TMS are [44]: to enable a problem solver to generate explanations for its conclusions; to help recover from inconsistencies by tracing back and finding the culprit; to avoid redundant computations by caching inferences that the problem-solver has already derived; to guide the backtracking using dependency-network of propositions; and to support nonmonotonic reasoning such that in view of new evidence the current set of beliefs can be revised.

There is a large body of research literature on different approaches to truth maintenance and its applications in a wide range of AI problem domains (see [110, 106]). The different approaches to TMS may be broadly classified as the justification-based truth maintenance systems (JTMS) (e.g., [37]), logic-based truth maintenance systems (LTMS) (e.g., [108, 109]), and assumption-based truth maintenance systems (ATMS) (e.g., [26, 147]). Chapter 4 gives a brief survey of these approaches to TMSs.

---

[4]Some TMSs insist on global consistency(e.g., [37]), others require only local consistency such that mutually incompatible conclusions are possible (e.g., ATMS [26]).

2.5 Conclusion

We have introduced various approaches to deal with efficiency of logic-based reasoning systems. Classification of these approaches is based on either weakening the representational and deductive capability of the logical framework, or looking for the better heuristics or techniques to improve empirical efficiency of the reasoning systems. While traditional complexity measures can be applied to deal with efficiency aspects of the logically weak reasoning systems, we do not intend to hamper the expressiveness and deductive power of first-order logic. However, one might think of using this approach from two points of view. The first, suggested in [61] and inspired by [162], is that if we can find a decidable subcomponent of a reasoning task, then the traditional complexity theory can be used to predict the computational behavior of that subcomponent. The second is to relax the completeness criteria to terminate the reasoning process when all resources are exhausted.

The representational adequacy and deductive capability of first-order logic provide the fundamental basis for intelligence systems, which should not be compromised just for the sake of tractability, especially at this crucial time when the disillusioned members of the AI community are trying to refuel the old debate of proceduralist versus logicist [117]. On the other side, heuristic-based techniques have been playing a significant role in the development of intelligent systems for more than the last two decades, and cannot be dismissed lightly. We have classified these techniques under the headings of meta-reasoning, constraint satisfaction and truth maintenance systems. While a great deal of overlap among the intuitions underlying each of these distinct approaches, the research in these fields is more or less self-contained. A major concern here is to integrate ideas from these distinct subfields of Artificial Intelligence, to promote understanding across subfields.

Having briefly discussed efficiency aspects of logic-based reasoning systems in general, the rest of this dissertation is dedicated exclusively to logic-based nonmonotonic reasoning systems. We consider nonmonotonic reasoning from the perspective of the THEORIST hypothetical reasoning framework [136, 134]. The next chapter describes the basic THEORIST framework and its extension to generalize most pragmatic techniques which in turn significantly improves the empirical efficiency of nonmonotonic reasoning.

Chapter 3

THEORIST: A Hypothetical Reasoning Framework

3.1 Introduction

In Artificial Intelligence, we have to deal with the fact that our knowledge is imperfect and is likely to remain so. Nevertheless, human intelligence makes decisions (or draws conclusions) based on the current beliefs using commonsense, and revises such conclusions in view of the contradictory information. Here we present a hypothetical reasoning framework called THEORIST [136, 134] that models commonsense reasoning, sometimes referred to as nonmonotonic reasoning, by maintaining potentially refutable theories.

THEORIST provides a practically useful and logically meaningful tool to model nonmonotonic reasoning as an instance of hypothetical reasoning. The basic idea is that nonmonotonic reasoning can be best viewed as a scientific theory formation process [141, 149, 73, 150, 79]. This involves not only observing and collecting facts about the domain of discourse, but also hypothesizing, constructing explanations (or making predictions), and testing and selecting (or revising) theories.

The next section describes the philosophical foundation of THEORIST (i.e., scientific theory formation). Section 3.3 presents the basic framework of THEORIST. Section 3.4 describes an extension to the basic architecture of THEORIST. Section 3.5 sketches the computational efficiency issues. Finally, we conclude this chapter.

3.2 Scientific Theory Formation

The question of how experiment and hypothesis could lead to knowledge has been a major concern in the philosophy of science. While the *empirist* position is that knowledge comes through sense experience, hypothetical reasoning emphasizes on the use of suppositions[1] to make predictions or to explain observations, rather than the derivation of laws from observations. These explanations are referred to as "scientific theories" and have been extensively studied in [141, 73, 150].

Scientific theory formation has long been a classical problem in the philosophy of science. Despite its wide and frequent use by philosophers and scientists, the term scientific theory or rule has been defined only in the form of a *code or rule of craft*. Popper [142] discarded the prevailing ambiguous definitions and suggested that scientific theories are conjectures or tentative hypotheses which may be rejected or accepted on the basis of observations. In other words, a scientific theory is a set of sentences or laws which form an *explanation* for an observation entailed by that theory along with facts. If the observation can be explained by a theory, the theory may be accepted; otherwise it is rejected.

Hempel's [73] *scientific explanation* or *deductive-nomological model* is based on the view that scientific theories are to be understood as sets of axioms in formal deductive systems. The views of both Hempel [73] and Popper [142] are *hypothetico-deductive*, but the former says that the theories are confirmed by deducing their consequences from the axioms, and checking to see whether the prediction holds; and the later says that the main role of prediction should be the attempt to falsify theories, not to confirm them.

Essentially, the aforementioned approach to scientific theory formation is contrary to the belief that science proceeds from observation to theory. Here, reasoning with pure observations alone, without a theory, is absurd as explained in [142, Page 47]:

> "The problem 'which comes first, the hypothesis (H) or the observation (O),' is soluble; as is the problem, 'which comes first, the hen (H) or the egg (O)'. The reply to the latter is, 'An earlier kind of egg'; to the former, 'An earlier kind of hypothesis'. It

[1] The propositions whose truth-status is undetermined or doubtful.

> is quite true that any particular hypothesis we choose will have been preceded by observations—the observation, ... If they are significant, if they created a need for explanation and thus gave rise to the invention of a hypothesis, it was because they could not be explained within the old theoretical framework, the old horizon of expectations."

In our approach, two kinds of sentences comprise the knowledge of an intelligent agent: those that we accept as true (i.e., facts) and those we tentatively treat as true (i.e., hypotheses). These facts and hypotheses are used to form a potential explanation for a deductively entailed conclusion. Scientifically speaking, a potential explanation can be subjected to crucial experiments. If an explanation passes the tests, then it is confirmed and gains strength. In Popper's words:

> "A scientist, whether theorist or experimenter, puts forward statements, and tests them step by step. In the field of the empirical sciences, more particularly, he constructs hypotheses, or systems of theories, and tests them against experience by observation or experiment." [141, page 27]

> "From a new idea, put tentatively, and not yet justified in any way—an anticipation, a hypothesis, a theoretical system, or what you will—conclusions are drawn by means of logical deduction. These conclusions are then compared with one another and with relevant statements, or as to find what logical relations (equivalence, derivability, compatibility, incompatibility) exist between them." [141, page 37]

Recently, Israel [79] argues that in order to capture nonmonotonic reasoning classical logic need not to be augmented with new inference rules, rather the reasoning process should be considered as scientific theory formation. Inspired from the Popper's philosophy [141, 142] and in the view of Israel's argument, Poole et al. proposed a logical framework for scientific theory formation—THEORIST. In attempting to deduce observation sentences, Theorist identifies instances of possible hypotheses as nomological explanations: consistent sets of hypothesis instances required to deduce observations [73]. The question 'Where do hypotheses come from?' is most difficult. Its answer is beyond the scope of this dissertation, however, the philosophical

literature [149, 150, 143] would help to understand some intuitions behind the THEORIST's possible hypotheses. Here, we must note that possible hypotheses in THEORIST are supplied by the user, such that THEORIST can use them to construct explanations.

3.3 The Basic Framework

The logical framework of THEORIST is based on distinguishing two kinds of sentences: those asserted as known to be true, and those we assert as tentatively true. The formal language of clausal logic L is used to represent knowledge about the world. The THEORIST knowledge base comprises of *facts* F, *possible hypotheses* H, *observations* O and *theories* T. The *facts* are a set of formulas of L that have the user's intended problem domain as a model. The *possible hypotheses* are a set of formulas of L which the user has classified as tentatively true. These formulas may be viewed as sentence schemas, instances of which may be used to augment the *facts* in a derivation. It is also possible to consider H as the set of all possible hypothesis instances ranging over the application domain's universe of discourse U, e.g., Prolog-based implementations consider U to be Herbrand universe of $F \cup H \cup O$. The consistency of hypotheses is tested by failing to prove the negation of a conclusion they support. In resolution-based clausal systems, this gives rise to the problem called reverse Skolemization, i.e., making the correct interpretation of the negation of sentences into which Skolem terms were previously introduced. The introduction of the Skolem functions may enlarge the domain U over which variables range, and thus require the special consideration of reverse Skolemization [139, 42].

Observations are a set of formulas of L, typically considered as goals or sentences requiring explanation, as in the expression

$$F \cup E \models O$$

E denotes any consistent set of hypothesis instances; it is called an ***explanation*** for O, as illustrated in Figure 2. Proof theoretically, the task is to construct a derivation of O from F, possibly using hypothesis instances.

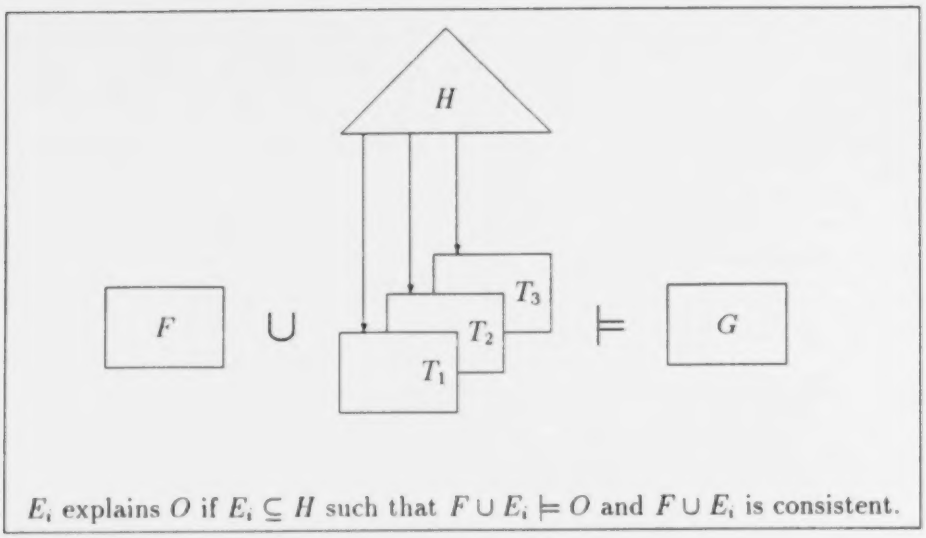


Figure 2: THEORIST

Definition 2 (explanation) *E is an explanation for O if* $\forall\, e \in E$, *e is an instance of hypothesis h, where* $h \in H$, *such that*
$F \cup E \models O$ *and* $F \cup E \not\models \neg e$[2].

Most of our implementations use a resolution-based full clausal logic, together with the following syntactic conventions for distinguishing facts and hypotheses:

fact *Clause* means that $Clause \in F$.
hypothesis *Clause* means that $Clause \in H$.

As an alternative to the latter specification of hypotheses, hypothesis *names* are introduced to provide a convenient method of recording the instantiation of arbitrary clauses classified as hypotheses (cf. [134]). The syntax

hypothesis *Name: Clause.*

[2] E is minimal, if there is no E' such that $F \cup E' \models O$ and $E' \subset E$

is equivalent to

fact *Clause* ← *Name*.
hypothesis *Name*.

To illustrate the process of explanation construction, we consider the standard example of *Tweety* the non-flying bird. The problem is represented as given in Example 3.1.

fact bird(tweety).
fact bird(X) ← emu(X).
fact ¬ flies(X) ← emu(X).
hypothesis birdsfly(X): flies(X) ← bird(X).

Example 3.1: Tweety the non-flying bird

When we ask the system to explain the observation $flies(tweety)$, it can be explained by using an instance of hypothesis $birdsfly(tweety)$ because $\neg birdsfly(tweety)$ cannot be derived from the given facts. However, we then learn that tweety is an emu and add this information to the knowledge-base as a new fact: **fact** emu(tweety). The system can no longer explain the observation $flies(tweety)$, because it is inconsistent to use the hypothesis instance $birdsfly(tweety)$. In fact, $\neg flies(tweety)$ is now a consequence of the facts.

The user interacts with Theorist by supplying facts and hypotheses, and by asking queries—requests for explanation or prediction. Figure 3 shows a typical dialogue with Theorist.

In Figure 3, [theorist]: is the system prompt and the boldface text indicates the user's input. In response to queries, Theorist provides answers (which provide variable binding) and theories. The theories are consistent sets of hypothesis instances. Each of them augments the facts to derive the query (i.e., an observation).

The Theorist project was started at the University of Waterloo by Poole, Goebel and Aleliunas in 1985. Some of the premier results on this project were published in [137, 59, 131, 136, 134]. Since then an enormous amount

[theorist]: **fact bird(tweety).**
[theorist]: **fact bird(X) ← emu(X).**
[theorist]: **fact ¬ flies(X) ← emu(X).**
[theorist]: **hypothesis birdsfly(X): flies(X) ← bird(X).**

[theorist]: **explain flies(tweety).**

Answer is: [flies(tweety)]
Theory is: [birdsfly(tweety)];

No (more) answers

[theorist]: **fact emu(tweety).**

[theorist]: **explain flies(tweety).**

No (more) answers

Figure 3: A Typical Interaction with Theorist

of research has been conducted to extend Theorist in many domains of reasoning. The original Theorist was mainly concerned with explaining observations. Later, it was extended to incorporate predictions [63, 133] and conjectures [138]. Recently, Brewka [13] introduced the idea of different modal categories (i.e., epistemic relevance or entrenchment as described in [149]) to classify the knowledge base into different levels of priorities. This prioritization then allows us to compute maximally preferred sub-theories. The possibility of using statistically founded knowledge in Theorist has been investigated by [126, 62]. The Theorist framework has been successfully applied in diagnostic reasoning [131, 59], analogical reasoning [80, 58, 185], planning [60, 64] and visual reasoning [48]. Investigating efficiency aspects of theorem proving in Theorist is the main goal of this thesis. The partial results have been published in [155, 154, 156, 161, 158, 160].

3.4 Extended-THEORIST

Since explanation in Theorist depends on logical satisfaction and consistency, the use of a logical language as powerful as first-order logic brings the prob-

lem of undecidability to the forefront. While undecidability is a worst case scenario in theory, it is not a usual case in many interesting problem domains. Apart from that fact, whether we encounter the problem of undecidability or not, all of the extensions or applications of the Theorist framework beg the issue of computational efficiency. Since it is highly desirable to have Theorist's underlying logic as powerful as first-order logic [72, 124], we must resort to heuristics; that is the only way we can pursue the efficiency aspects of such a powerful reasoning system.

For the framework defined in previous section, our implementation attempts to establish the consistency of a hypothesis instance by failing to find a proof for its negation (using facts and hypothesis instances already shown to be consistent). Note that, while checking consistency, we are not allowed to assume a hypothesis instance which is not already an element of a partially constructed explanation.

We extend this mechanism to consider new instances of hypotheses. In other words, we extend the computation to consider combinations of inconsistent hypothesis instances. The inconsistent set of hypothesis instances may be viewed as *nogoods* in terms of Doyle's truth maintenance system (TMS) [37].

In the definition below, T refers to a "theory" or working set of hypothesis instances not necessarily established as an explanation. T' is any set of instances of hypotheses not including the elements of T (i.e., $T \cap T' = \phi$). The extended proof tree is referred to as a consistency tree, which is a success subtree of the derivation tree for $F \cup T \cup T' \cup \{\neg t\}$ similar to the AND/OR tree of the MESON system (cf. [100, ch. 6]). The branches of the tree corresponding to successful derivations are called *success branches* (i.e., branches having facts or instances of hypotheses from $T \cup T'$ leading to empty leaves), branches corresponding to infinite derivations are called *infinite branches* (e.g., equivalence relationship between two literals may lead to a cyclic occurrence of the literals) and branches corresponding to failed derivations are called *failure branches*. A success branch may be viewed as a *goal tree* of the MESON proof procedure. For consistency trees, failure branches can be extended to success branches, by considering relevant instances of unverified hypotheses.

Our definition of consistency tree uses the terminology from [100]. A goal is a *parent goal* of its immediate subgoals. Subgoals are *successors* of their parent goals. The parent goal of a goal B is an *ancestor* of B and all of B's

descendants. A subgoal B is a descendent of goal A iff A is an ancestor of B. The antecedents of an implication used to create a new set of subgoals (successors) is called a *conjunctive* set of subgoals. Each element of this set is a *partner goal* for other elements. The sets of possible conjunctive sets of subgoals of a given goal form a disjunction of conjunctive sets. The ancestor partner goal is a partner of an ancestor of a given goal. A *substitution* θ is a finite set of the form $\{v_1/t_1, ..., v_n/t_n\}$, where each v_i is a variable, each t_i is a term distinct from v_i and the variables $v_1, ..., v_n$ are distinct. If S is a finite set of literals, then a substitution θ is called a *unifier* if $S\theta$ is a singleton. A unifier θ is called a *most general unifier* for S, if for each unifier σ of S, there exists a substitution γ such that $\sigma = \theta\gamma$.

The input to the following procedure are sentences of the form $L \leftarrow L_1 \wedge ... \wedge L_n$, where L are literals. The procedure then uses all general contrapositives of this implication while constructing a rooted tree for the goal sentence. For example, the implication $P \wedge \neg Q \rightarrow R$, where P, Q and R are atoms, has the general contrapositives:

$\neg R \wedge \neg Q \rightarrow \neg P$; and
$P \wedge \neg R \rightarrow Q$.

Definition 3 (consistency tree) *Suppose g is an observation, F is a set of facts, T is a set of instances of possible hypotheses shown to be consistent and t is an instance of hypothesis such that $F \cup T \cup t \vdash g$. A* consistency tree *for t is a* success subtree *of the AND/OR derivation tree for $F \cup T \cup T' \cup \{\neg t\}$, where T' is a set of instances of possible hypotheses such that $T \cap T' = \phi$, and is defined as follows:*

- *$\neg t$, is a goal which is the root.*
- *Each node of a tree is a conjunction of possibly empty subgoals.*
- *If $\leftarrow A_1, .., A_i, ..., A_n$ is a goal and a literal A_i is selected by some computation rule, then for each input fact $A \leftarrow B_1, ..., B_k$ such that A is unifiable with A_i, the successor is*

$$\leftarrow (A_1, ..., A_{i-1}, B_1, .., B_k, A_{i+1}, ..., A_n)\theta$$

  where θ is a most general unifier *of A_i and A.*
- *If A is a unit goal and $\neg A$ is an ancestor of A, then A has no descendant (by* reductio ad absurdum*).*

- *If A is a unit goal and $A \in T$ or $A \in T'$, then A has no descendant.*
- *Nodes which are empty goals have no descendants.*

fact $o \leftarrow a \wedge b$.	
fact $o \leftarrow \neg a \wedge f$.	
fact f.	**hypothesis** t_1.
fact $a \leftarrow f \wedge t_1$.	**hypothesis** t_2.
fact $\neg a \leftarrow b \wedge t_2$.	**hypothesis** t_3.
fact $b \leftarrow f \wedge t_3$.	

Example 3.2: Theorist Database

The idea of a consistency tree may be illustrated by considering the Theorist database shown in Example 3.2. To explain an observation o, theorist can construct proofs of the two subgoals a and b (it could be $\neg a$ and f too). If a is selected first, its proof would require hypothesis instance t_1. Before using t_1 in the proof tree, the theorem prover has to establish the unprovability of $\neg t_1$. If t_1 is consistent, then there is no success branch of the derivation tree constructed from the facts and $\{\neg t_1\}$. Figure 4 (a) shows one of the finitely failed branches. However, if we use new hypothesis instances to extend the failure branches, some of them become success branches. We call these extended proof trees consistency trees. Figure 4 (b) shows a consistency tree for t_1 constructed using Definition 3. Each of its nodes represents an atomic goal.

The main goal of constructing consistency trees is to identify inconsistent sets of hypothesis instances. These inconsistent sets can then be recorded to avoid future futile search. Figure 4 (b) shows that $F \models t_3 \wedge t_2 \rightarrow \neg t_1$ (i.e., $F \cup t_1 \wedge t_2 \wedge t_3 \vdash false$); $\{t_1, t_2, t_3\}$ is an inconsistent set of hypothesis instances. We can use this information so that all supersets of such sets are eliminated from the search space.

In the subsequent chapters, we will show how this simple idea of consistency tree can be used to generalize most of the pragmatic approaches to efficient reasoning.

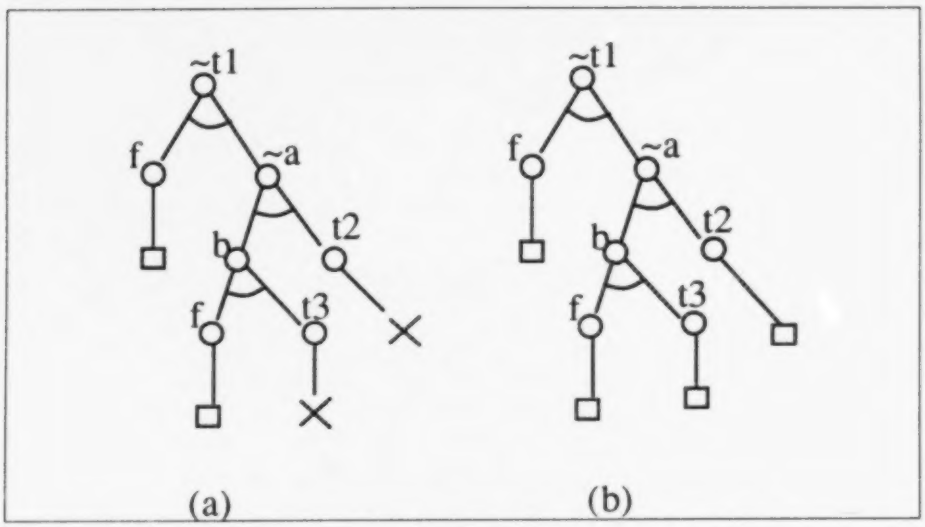


Figure 4: Consistency tree $tree(\neg t_1)$

3.5 The Efficiency Issues in THEORIST

THEORIST may be viewed as a theorem prover for full first-order logic where a deductive derivation of a goal formula, called an observation, is constructed by augmenting facts with consistent instances of hypotheses. These consistent hypothesis instances are maintained as potentially refutable theories that provides the basis for nonmonotonicity. When THEORIST is asked to solve reasoning problems, our major concern is how efficiently it generates solutions,[3] especially in the nonmonotonic reasoning environment where we have to deal with multiple solutions (possibly incompatible). So, we consider efficiency of THEORIST from the perspective of the following issues:

- Efficiently searching for an explanation for a given set of observations, i.e., finding a solution as quickly as possible, in the complex

[3]Since the traditional complexity measures are void for an expressively rich system such as THEORIST, the overall computational efficiency can only be addressed from the empirical view point.

search space. For example, introduce a caching mechanism to avoid re-derivation of the same proof or explanation trees, and *a priori* identify inconsistent set of hypothesis instances that can be ignored as part of any potential explanation.

- Often, we need to compute more than one explanation. The total number of explanations may be exponentially large. Thus, reducing the space of possible explanations[4] would significantly improve efficiency of the underlying reasoning system.

- Distinguishing between competing explanatory theories using some rational epistemic policies such as generating tests to select the unrefuted theories and eliminating those which are refuted. If we employ such a discrimination procedure incrementally, it will help not only in selecting the best possible alternate, but also in reducing the total number of possible explanations.

3.6 Conclusion

A number of empirical techniques have been developed for the last two decades, as briefly surveyed in the previous chapter, for reducing the search space. Most of the pragmatic or heuristic-based search techniques may be characterized under the three headings: constraint satisfaction—specifying the reasoning task as a system of constraints and then using constraints to *a priori* reduce the search space; meta-reasoning—specifying control at the meta-level to influence the search of deduction at the object-level; and truth maintenance systems—recording dependencies among the propositions and possible inconsistencies as nogoods to avoid the redundant computation and future futile search in the inconsistent space. These approaches appear to be distinct but have common goals and similar intuitions.

Also, quite a few number of domain dependent theory selection/preference approaches have been invented in the area of nonmonotonic reasoning. However, it is debatable what criteria should be used for this kind of classification.

---

[4]Many of them are not relevant or correct so constructing them is a futile activity, e.g., a diagnosis of pregnancy for a male patient.

In this dissertation, we propose a method based on identifying crucial experiments to discriminate between two mutually incompatible theories.

The ultimate quest, of course, should be to discover a single uniform framework embracing all the seemingly different search techniques as well as the theory preference techniques. Subsequent chapters describe how the *Extended-Theorist* is an attempt to achieve a uniform framework for problem solving based on hypothetical reasoning.

Chapter 4

Extended-Theorist: A Truth Maintenance System

4.1 Introduction

The fundamental achievement of truth maintenance systems (TMSs) has been the implementation of an relatively efficient problem solving support system by employing some sort of caching mechanism to record the derived implicit knowledge and by introducing a policy to revise a system's beliefs in the view of contradictory information. The major limitations of these TMSs are: the lack of expressiveness; then allow only restricted propositional clauses, (e.g., TMS [37], ATMS [26]); some of them do not capture nonmonotonic reasoning (e.g., ATMS [26], LTMS[109]); the inability to deal with multiple contexts due to the requirement for global consistency of knowledge base (e.g., TMS [37]); and those deal with multiple contexts simultaneously require compilation of all inconsistencies and label for each propositional literal (e.g., ATMS [26]). In the view of the pros and the cons of the traditional TMSs, what we need is a more general framework for truth maintenance that can provide an empirically efficient as well as expressively rich problem solving support system.

To achieve efficiency of problem solving in Theorist, we extend the basic framework by introducing a truth maintenance component, supported on a collection of truth tables, and by modifying the procedures for explanation generation. Unlike other approaches to truth maintenance, we are seeking

a first-order logic based framework for efficient problem solving with the following main features:

- **Expressive power:** The traditional TMSs use relatively weak logical language (e.g., restricted propositional clauses) that is inadequate to represent a real world problem. So, one concern is a logical language with more expressive power.

- **Generality:** The problem solving system must be suitable for efficiently searching a single solution as well as multiple solutions. It should also be able to deal with mutually contradictory solutions (e.g., incompatible diagnoses).

- **Nonmonotonicity:** The derived conclusions must be tentative or potentially refutable such that the new evidence may rescind old beliefs.

- **Inconsistencies:** Inconsistent combinations of hypothesis instances (i.e., nogoods) should be incrementally identified while testing consistency of hypothesis instances rather than compiling all of them.

- **Distinguishing competing theories:** Most TMSs are obliged to find one context or generate one explanation, however, ATMS simultaneously maintains all contexts. The total number of contexts may be exponentially large. Although we would like to have more than one solution, computing an exponentially large solution space is not at all desirable. What we need is an incremental mechanism to distinguish between incompatible theories, and help to keep the solution space manageable.

- **Avoiding redundant computation:** The problem solver should be able to record the derived implicit knowledge as explicit facts, so that re-derivation of those facts can be avoided by simply looking into the truth maintenance tables.

- **Update:** The truth maintenance tables must be dynamically updated in the light of new derived information.

The rest of this chapter is organized in several sections. Section 4.2 reviews the traditional TMSs. Section 4.3 provides logical specification of truth

maintenance in hypothetical reasoning, which includes definitions and properties of the truth maintenance tables, incremental algorithms for generating explanations, and update procedures for all truth maintenance tables. Also, we discuss efficiency and implementation issues. Section 4.4 discusses a relationship with other TMSs with respect to their advantages and limitations. Finally, we summarize our contributions.

4.2 Background

The development of truth maintenance systems (TMSs) has been an active research area since the late seventies. The idea is that the problem-solving system must be viewed as combination of two components (see Figure 5): the inference engine, which manipulates the knowledge about a specific problem domain using some rules of inference, and the truth maintenance system, which records dependencies among propositions and maintains a consistent view[1] of the knowledge base. The underlying intuition behind most TMSs is that while searching for a solution the problem-solver explores many states which are distinct but very similar. The obvious question is how to make use of these similarities such that the same state is not revisited and the inconsistencies once identified should not be repeated.

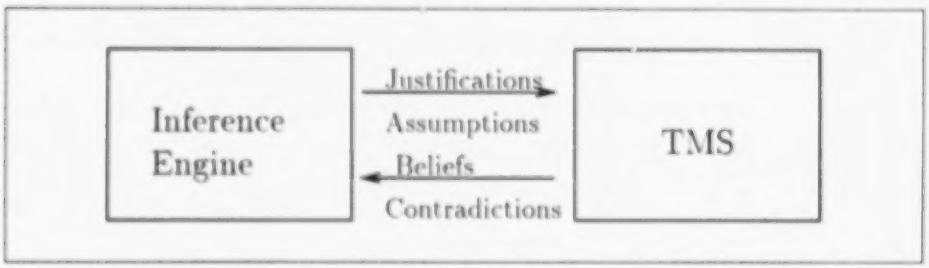


Figure 5: Problem-solver = Inference Engine + TMS

The main goals of a typical TMS are [44]: to enable a problem solver to generate explanations for its conclusions; to help recovering from inconsistencies by tracing back and finding the culprit; to avoid redundant computations

[1]Some TMSs insist on the global consistency(e.g., [37]), others require only local consistency such that mutually incompatible conclusions are possible (e.g., ATMS [26])

by caching inferences that the problem-solver has already derived; to guide the backtracking using a dependency-network of propositions; and to support nonmonotonic reasoning such that, in the view of new evidence, the current set of beliefs can be revised. There is a large body of research literature on different approaches to truth maintenance and its applications in a wide range of AI problem domains (see [110, 106]).

Doyle's truth maintenance system (TMS) [36, 37] is the first TMS where the dependencies among propositions and the dependency-directed backtracking modules are isolated from the inference engine for a problem-solver. McAllester [108, 109] introduced a logic-based approach to truth maintenance system known as LTMS. The LTMS basically allows full propositional clauses (cf. definite clauses in Doyle's TMS) to be as input, thus overcoming the expressive limitations of the propositional calculus in the classical TMS [36, 37]. Also, the LTMS avoids repeated discovery of the same contradictions—improving backtrack search. However, the LTMS does not support nonmonotonicity. Later, de Kleer [26] introduced an assumption-based truth maintenance system (ATMS) that overcomes many limitations of the previous truth maintenance systems. The basic ATMS lacks nonmonotonicity which is later addressed in various ATMS extensions (e.g., [38]).

The different approaches to TMS may be broadly classified as the justification-based truth maintenance systems (JTMS) (e.g., [37]), logic-based truth maintenance systems (LTMS) (e.g., [108, 109]), and assumption-based truth maintenance systems (ATMS) (e.g., [26, 147]). In this section, we summarize[2] each of the three classes of TMSs. The next section gives full account of our approach, which is based on hypothetical reasoning, to truth maintenance and how it improves efficiency of problem-solving.

4.2.1 Justification-based TMS

The Justification-based TMS is the simplest and most commonly used class of truth maintenance systems (e.g., JTMS [44], TMS [37]). In JTMS [44], the truth maintenance component of the problem-solver records justifications in terms of dependencies between antecedents and consequents. These justifications then help the reasoner to efficiently find a solution. A data-structure

[2]The summary contains excerpts from [37, 108, 109, 26, 147, 44].

called a *node* is associated with the TMS. Each node represents a *premise*, or *contradiction* or an *assumption*:

- A node is a *premise* if it is explicitly indicated by the reasoner that it holds universally.
- A node is a *contradiction* if the reasoner has indicated that it can never hold.
- A node is an *assumption* if it is explicitly indicated belief which may be changed in future.

The JTMS is basically concerned with the set of justifications and the set of enabled assumptions. An *assumption* is enabled if the TMS believes it, however, which might be retracted later. A *justification* represents the inference steps used to derive a node from the combination of other nodes. A node is believed if it has at least one valid justification. The set of justifications grows monotonically, where as the set of assumptions is dynamically changing over time.

To specify the architecture of JTMS, we can consider each node as a propositional symbol. A subset of these symbols is characterized as a set of assumptions, say $\mathcal{A}$. Each justification is a propositional clause. If a set of such clauses is $\mathcal{J}$, then fundamental task of JTMS is to answer queries about whether a particular node, say n, logically follows from the sets $\mathcal{A}$ and $\mathcal{J}$. A node n is believed if $\mathcal{J} \cup \mathcal{A} \vdash n$, otherwise it is not believed.

A justification consists of three parts:

- *consequent*: the inferred node.
- *antecedents*: the nodes from which the consequent was inferred.
- *informant*: supplied by the reasoner to explain the consequent.

The Doyle's justification-based TMS [37] supports nonmonotonic inferences (cf. JTMS [44]). The two forms of justifications are used: support-list justification and conditional-proof justification.

Support-list justification:
(AND (IN < *inlist* >) (OUT < *outlist* >)).
Conditional-proof justification:
(CP < *consequeent* > < *inhypotheses* > < *outhypotheses* >).

A support-list justification is valid if each node in its *inlist* is believed, and each node in its *outlist* is not believed. The inference mechanism is nonmonotonic as a conclusion depends not only on what assumptions are believed to be true, but also what assumptions are not believed to be true. The conditional-proof justification represents an implication, a node derived by conditional proof of the consequent node from the hypothesis nodes. It is valid if each node of the *inhypotheses* is believed and each node of *outhypotheses* is not believed.

The truth maintenance component of the TMS caches every justification that helps in determining whether a node is believed or not, by looking for a a valid justification. The convention of recording justifications for deductions allows a problem-solver to generate explanations for conclusions. Whenever a contradiction is encountered, the records of dependencies are considered to determine the cause of the contradiction. Then the culprit is removed to ensure consistency of the knowledge base. This mechanism is called dependency-directed backtracking. The set of assumptions contributing to contradiction is called a *nogood* set. The TMS process may be illustrated using a simple example [37] of scheduling a meeting.

The meeting is preferred to be held at 1:00pm in either room GSB611 or GSB619.

$$\begin{array}{ll} 1 & meeting_time = 1:00pm \quad (SL\ ()\ (2)) \\ 2 & meeting_time \neq 1:00pm \\ 3 & meeting_room = GSB619 \quad (SL\ ()\ (4)) \\ 4 & meeting_room = GSB611 \end{array}$$

Initially, the TMS makes node 1 and 3 IN (believed) and other two OUT (not believed). Now suppose this combination of time and room is not possible as some other meeting was previously scheduled in the same room at this time. Then the TMS creates a new node 5 and declares it as a contradiction:

$$5 \quad contradiction \quad (SL\ (1\ 3)\ ())$$

The dependency-directed backtracking process looks for reasons of this contradiction, and then creates a new node 6, that is labeled as *nogood*. The node 3 is arbitrarily blamed for this contradiction, which in turn requires justification of node 4 which is otherwise not to be believed:

6	*nogood* 1 3	$(CP\ 5\ (1\ 3)\ ())$
4	$meeting_room = GSB611$	$(SL\ (6\ 1)\ ())$

Finally, the TMS labels the nodes 1, 4 & 6 to be IN (i.e., believed) and the nodes 2, 3 & 5 to be OUT (i.e., not believed).

4.2.2 Logic-based TMS

The Logic-based truth maintenance system (LTMS) [108, 44] is a generalization of JTMS. The basic difference is that the LTMS allows an arbitrary propositional clause as input. While justifications in JTMS are simply definite clauses (i.e., $x_1 \wedge ... \wedge \Rightarrow n$ where x_i and n are all TMS nodes), the LTMS improves the expressive power that results in significant problem solving advantages. Suppose we want to express a clause such $a \vee b \vee c$. The JTMS needs six nodes and six justifications in the dependency network, where as the LTMS can do the same job with three nodes and one propositional clause. The definite clause $x \Rightarrow y$ allows JTMS to infer y from x, but it can not infer $\neg x$ from $\neg y$. In addition, the LTMS improves backtrack search by avoiding repeated occurrence of the same contradictions.

Similar to the JTMS, a *node* is the basic data structure in the LTMS for representing propositional symbols. While JTMS allows only two kinds of label for each TMS node such as IN and OUT, the LTMS allows three types of labels such as $TRUE$, $FALSE$ and $UNKNOWN$. Suppose $\mathcal{C}$ is a set of clauses supplied by the inference engine. A node is labeled $TRUE$ if it logically follows from $\mathcal{C}$ and the set of enabled assumptions. A node is labeled $FALSE$ if its negation is derivable from $\mathcal{C}$, and labeled $UNKNOWN$ otherwise.

Let us consider $\mathcal{S}$ be a set of propositional symbols associated with each LTMS node. A subset of $\mathcal{S}$, say $\mathcal{A}$, is a set of assumption literals which are initially labeled either $TRUE$ or $FALSE$. A literal $x \in \mathcal{A}$ if x is labeled as $TRUE$ and $\neg x \in \mathcal{A}$ if x is labeled $FALSE$. The set $\mathcal{C}$ monotonically grows, while the size of set $\mathcal{A}$ dynamically changes depending on assertions and retractions. The main tasks of the LTMS are: to compute a label for a node (label is $TRUE$ or $FALSE$ or $UNKNOWN$ depending whether a propositional symbol associated with that node or its negation is derivable) to generate explanations for the labels; and to identify contradictions.

The explanation generation process in the LTMS can be illustrated with a problem of boolean constraint propagation. Suppose the input clauses are:

$$x \vee y$$
$$\neg y \vee z$$
$$\neg z \vee r$$

A node's label is explained using a sequence of steps say $S_1, ..., S_k$ where each S_i consists of a tuple of the form:

$$step\ no. \quad < conclusion > \quad < antecendent > \quad reason$$

If x is assumed to be $FALSE$, then the following steps would be required to generate an explanation for r:

step	$< conclusion >$	$< antecedent >$	*reason*
1	$\neg x$	$\{\}$	*Assumption*
2	y	$\{1\}$	$x \vee y$
3	z	$\{2\}$	$\neg y \vee z$
4	r	$\{3\}$	$\neg z \vee r$

A node can appear in multiple clauses, so there can be a very large number of possible explanations for any particular node. The LTMS is obliged to find only one of them.

4.2.3 Assumption-based TMS

The assumption-based truth maintenance system (ATMS) (e.g., [26, 44]) is basically an extension of the ideas presented in the justification-based TMS [37]. Its primary motivation is to avoid some of the main limitations of the JTMS such as the inability to simultaneously deal with multiple contexts. These multiple contexts may be mutually contradictory as well. The classical TMSs (both JTMS and LTMS) can change contexts by enabling and retracting assumptions. However, each such change requires relabeling of the TMS nodes that is very expensive if the rate of change of assumptions is higher than the number of queries being asked.

Similar to the other TMSs, a data-structure called a node is used. The ATMS node represents a *premise* if it is provided as a justification with no

antecedents. A node is an *assumption* if the reasoner wishes to make use of this node to derive other goals. A set of assumptions is called an *environment*. A node is *contradiction* if the reasoner concluded that it can never hold. A node is said to hold in an environment if it can be derived from that environment and the current set of justifications. An environment supporting a *contradiction* node is called an *inconsistent environment* or a *nogood*. An ATMS *justification* corresponds to the logical notion of derivation: a node is justified by a selected set of other nodes. A *context* is the set formed by the assumptions of a consistent environment combined with all nodes derivable from those assumptions.

A problem domain involving m assumptions has 2^m possible environments, and at most 2^m contexts. The assumption space of size 2^m is called the environment lattice. When a set of assumptions is identified to be inconsistent (i.e., *nogood*), all supersets of that set are removed from the environment lattice. Figure 6(a) indicates an environment lattice of three assumptions **A**, **B** & **C**. Figure 6(b) shows that if $\{\mathbf{A}\}$ is inconsistent (i.e., nogood) then all dark nodes (i.e., supersets of $\{\mathbf{A}\}$) become nogoods as well. Using *nogoods* to eliminate a large number of inconsistent environments is a key to the search efficiency of the ATMS.

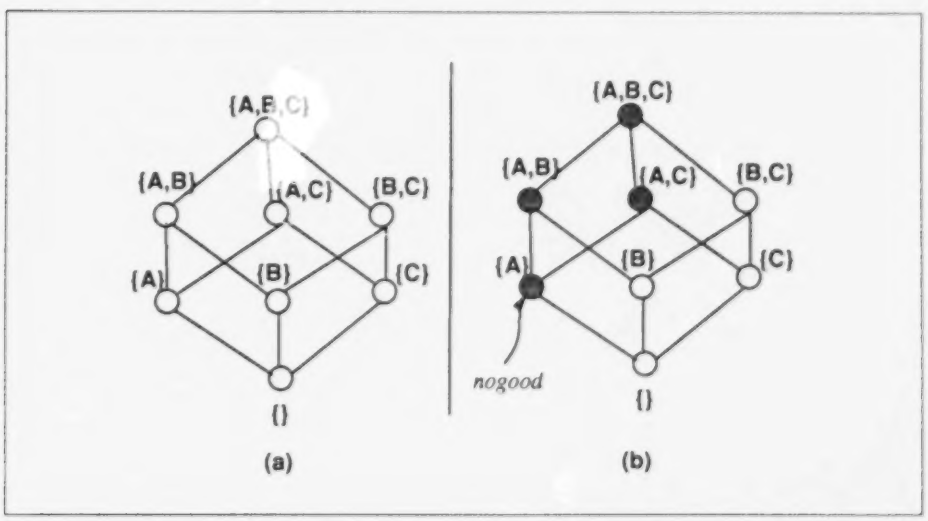


Figure 6: Environment Lattice

Logically the ATMS may be considered as a propositional theorem prover. The set of ATMS nodes define a set of propositional symbols, say $\mathcal{S}$. A subset $\mathcal{A} \subset \mathcal{S}$ is marked a set of assumptions. Similar to the JTMS, each justification in ATMS is a definite propositional clause. Suppose $\mathcal{C}$ is a set of definite clauses, and E is an environment (i.e., $E \subset \mathcal{A}$). Then a node n is said to hold in environment E, if n can be propositionally derived from $E \cup \mathcal{C}$. The ATMS incrementally entertains a stream of additional nodes, justifications, and assumptions by maintaining a label (a set of environments) for each node. If $\mathcal{L} = \{E_1, ..., E_k\}$ is a label of the node n, then each $E_i \in \mathcal{L}$ supports the node n (*soundness*); E_i is not a nogood (*consistency*); every consistent environment E supporting n is a superset of E_i (*completeness*); and E_i is a minimal set w.r.t. set inclusion (*minimality*). Let us consider an example to illustrate how the ATMS functions:

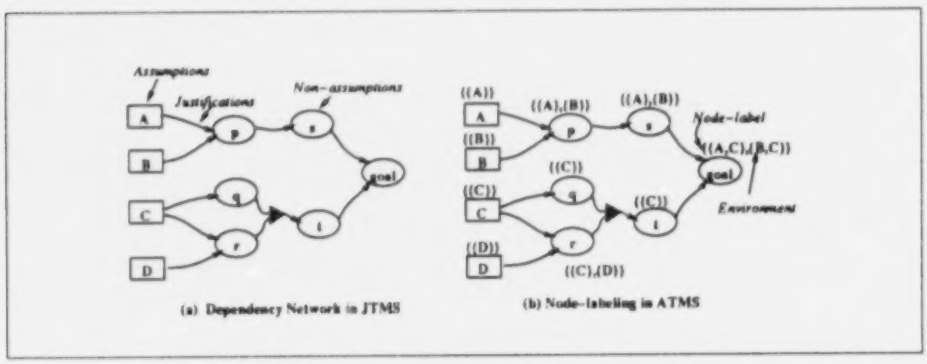


Figure 7: Comparing the JTMS and the ATMS nodes

Suppose we have a set of symbols $\mathcal{S} = \{A, B, C, D, p, q, r, s, t, goal\}$ where **A,B,C & D** are assumptions. Let us consider the following relationships among these propositions (see Figure 7):

$$
\begin{array}{llll}
A \rightarrow p & p \rightarrow s & & s \rightarrow goal \\
B \rightarrow p & q \wedge r \rightarrow t & & t \rightarrow goal \\
C \rightarrow q & & & \\
C \rightarrow r & & & \\
D \rightarrow r & & &
\end{array}
$$

Figure 7(b) shows a label for each ATMS node. Each label is a set of environments. To answer the query whether a given proposition is derivable from the given set of clauses, the ATMS looks for a consistent environment in the label of that proposition. In Figure 7 (b), the proposition *goal* holds in both environment {**A**,**C**} & {**B**,**C**}. These consistent environments together with the nodes they support form *contexts*. However, if the ATMS learns a new fact that the environment {**A**} supports a contradiction, then it becomes nogood and all of its supersets are eliminated from the environment lattice, reducing possible contexts. Consequently the node *goal* now holds in only one environment {**B**,**C**}.

4.3 Hypothetical Reasoning-based TMS

Our motivation to develop a hypothetical reasoning-based TMS is, in part, due to simplicity, novelty, and expressive richness of Theorist [136, 134]. Theorist captures nonmonotonic reasoning by maintaining potentially refutable theories, known as scientific theories, and allows first-order logic to be used as a representation language. While conventional theorem proving techniques answer queries by showing whether the given formula is a theorem or not, Theorist attempts to deduce observation sentences by identifying instances of hypotheses (i.e., assumptions) as nomological explanations.

To improve efficiency of theorem proving, the ATMS [26] compiles a set of possible environments (called a label) for each node that forms all possible contexts for the proposition represented by that node. Logically, computing ATMS labels may be viewed as compiling the knowledge base (a set of propositional clauses) into *prime implicants* [147]. A prime implicant of a set of clauses Σ is a set PI where $\Sigma \models PI$ and there is no $PI' \subset PI$ such that $\Sigma \models PI'$. When the prime implicants are recorded as explicit beliefs, queries can be answered efficiently.

However, there are two reasons for not adopting the ATMS style of efficient theorem proving in Theorist: first, the preciseness of an expressively rich logical language makes a large body of our knowledge *implicit*; and second, even if we restrict the language to be decidable, the tractability of reasoning cannot be guaranteed—compilation of all prime implicants may not be effectively computable. Thus, we cannot think of compiling all implicit knowledge in Theorist. Rather, our approach is based on a middle

ground that the implicit beliefs[3] can be incrementally derived, as needed, and recorded and used in the future computations. The incremental compilation approach is not only the best possible way to deal with the practical efficiency of reasoning, but also makes sense irrespective of decidability or undecidability of the underlying logical system.

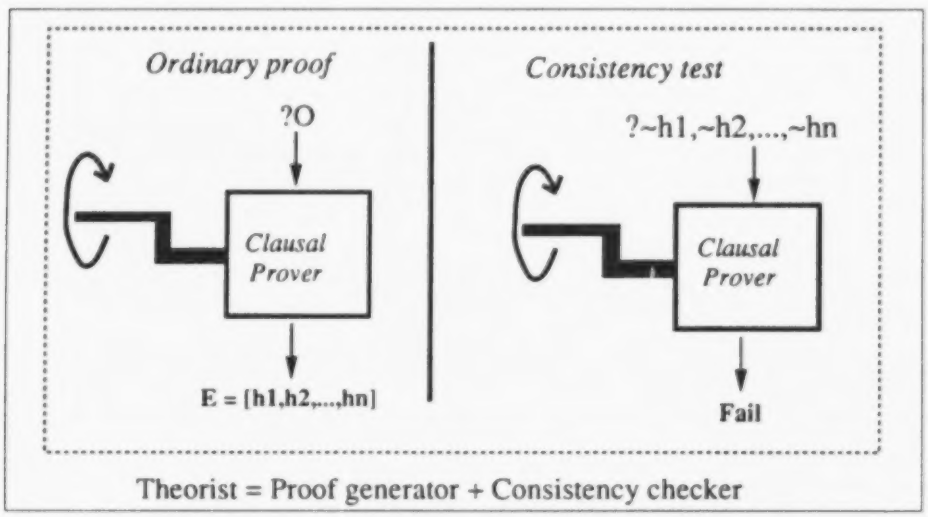


Figure 8: Explanation construction in Theorist

4.3.1 The basic architecture

The basic architecture of Theorist is described in [136, 134]. Figure 8 indicates how the basic Theorist framework is implemented, where the ordinary *proof generator* attempts to derive the goal formula by extending facts with possible hypotheses. Whenever an hypothesis instance is used by the proof generator, it requests the *consistency checker* to test its consistency by failing to prove the negation of that hypothesis instance.

We extend the basic architecture by introducing a truth maintenance component (see Figure 9), which is a collection of truth maintenance tables,

[3]Here, beliefs and knowledge are meant to be the same.

and by modifying the *proof generator* and *consistency checker*. The *new proof generator* constructs ordinary proofs by consulting truth maintenance tables to avoid redundant computations of the already derived facts or explanations or inconsistencies (i.e., nogoods). Similarly, the *new consistency checker* uses the truth maintenance tables. In return, both the *new proof generator* and *new consistency checker* provides useful information to the truth maintenance tables.

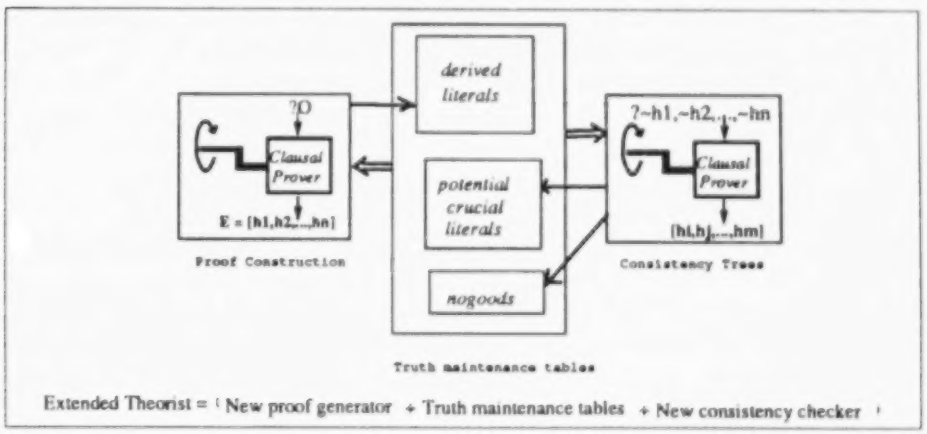


Figure 9: Explanation generation in Extended-Theorist

4.3.2 Logical Specification

The Theorist hypothetical reasoning framework [136, 134] views reasoning as constructing theories that logically imply a set of observations. The representation language, full first-order clausal logic, consists of two kinds of statements: *facts* and *possible hypotheses*. We accept facts as true statements just like ordinary assertions in a logical theory. Possible hypotheses, however, are given a different epistemological status than facts (cf. *assumptions* in the ATMS [26]). They are used as a kind of schema for axioms and viewed as the specification for the tentatively true knowledge that is essential for extending facts to generate explanations (cf. generating contexts in the ATMS [26]) for a set of observations (i.e., queries in the traditional TMSs).

Like traditional TMSs, the fundamental task here is to answer a wide

variety of queries whether a given formula about the problem domain is explainable or not. Let F be a set of facts and H be a set of possible hypotheses. Suppose $\mathcal{Q}$ is a query. Then Theorist answers 'yes', iff

$F \cup \mathcal{E} \models \mathcal{Q}$, and
$F \cup \{\mathcal{E} - e_i\} \not\models \neg e_i$ for each $e_i \in \mathcal{E}$.

Otherwise, the answer is 'no.' The set $\mathcal{E}$ consists of hypothesis instances drawn from H. $\mathcal{E}$ is said to be an *explanation* for the query $\mathcal{Q}$. Here, $F \cup \mathcal{E}$ is an ordinary logical theory having standard Tarskian semantics.

While constructing $\mathcal{E}$, the overall theorem proving task is divided into two parts: ordinary proof construction for a query, and performing consistency test for each hypothesis instance used in $\mathcal{E}$. Essentially, each of these two components is a clausal theorem prover. As shown in Figure 9, each clausal prover is extended to supply useful information to the truth maintenance tables and then use that information for future derivations.

A set $\mathcal{N}$ of hypothesis instances drawn from H is inconsistent iff

$F \cup \mathcal{N} \vdash \perp$.

$\mathcal{N}$ is referred to as a *nogood*. Note that all supersets of a nogood are nogoods.

For each explanation $\mathcal{E}$ for a query $\mathcal{Q}$, we define the following properties:

- each explanation $\mathcal{E}$ and facts F entails $\mathcal{Q}$.
- if $\mathcal{E}$ is empty, then $\mathcal{Q}$ is derivable exclusively from the facts.
- an explanation $\mathcal{E}$ is *minimal* if there is no explanation $\mathcal{E}'$, such that $\mathcal{E} \subset \mathcal{E}'$.[4]
- there is no explanation $\mathcal{E}$ which is a nogood.
- the two explanations $\mathcal{E}_i$ and $\mathcal{E}_j$ are mutually incompatible if $\mathcal{E}_i \cup \mathcal{E}_j$ is a nogood.
- a literal C is a crucial literal between two incompatible explanations $\mathcal{E}_i$ and $\mathcal{E}_j$ iff $F \cup \mathcal{E}_i \models C$ and $F \cup \mathcal{E}_j \models \neg C$, thus, the truth value of C will eliminate one of the two incompatible explanations.

[4]Here, the minimality is considered w.r.t. set inclusion. However, different forms of *minimality*, e.g., least presumptive explanation, (see [135]) could be considered.

Here, a query may be an arbitrary formula rather than a single literal (cf., ATMS which allows only a single literal [147]). We do not maintain all possible explanations for a given query. These explanations are generated incrementally as needed. The set of nogoods monotonically increases, which in turn reduces the solution space.

4.3.3 Truth Maintenance Tables

While constructing explanations for a given query, the Theorist hypothetical reasoning system is dealing with derivations from facts; the literals that can be derived by facts and hypotheses instances; the literals whose derivations depend not only on the inclusion of some sets of hypotheses instances but also on the exclusion of certain sets of hypotheses instances; and the sets of hypotheses instances that lead to contradictions. The truth maintenance component (see Figure 9) basically attempts to record "useful" derivations as much as possible. Here, a "useful derivation" means that the particular information can be used to improve the search in the future computations.

In Theorist, there are two kinds of clausal provers involved—the ordinary proof generator and the consistency checker. Both the *proof generator* and the *consistency checker* are modified to incrementally supply the useful information to the truth maintenance tables, and then use the same information to improve their search efficiency.[5]

The truth maintenance component consists of three kinds of tables: a table of *derived literals*, a table of *potential crucial literals* and a table of minimal *nogoods*. *Derived literals* are useful derivations extracted from the proof generator. *Potential crucial literals* and *nogoods* extracted from the consistency checker.

---

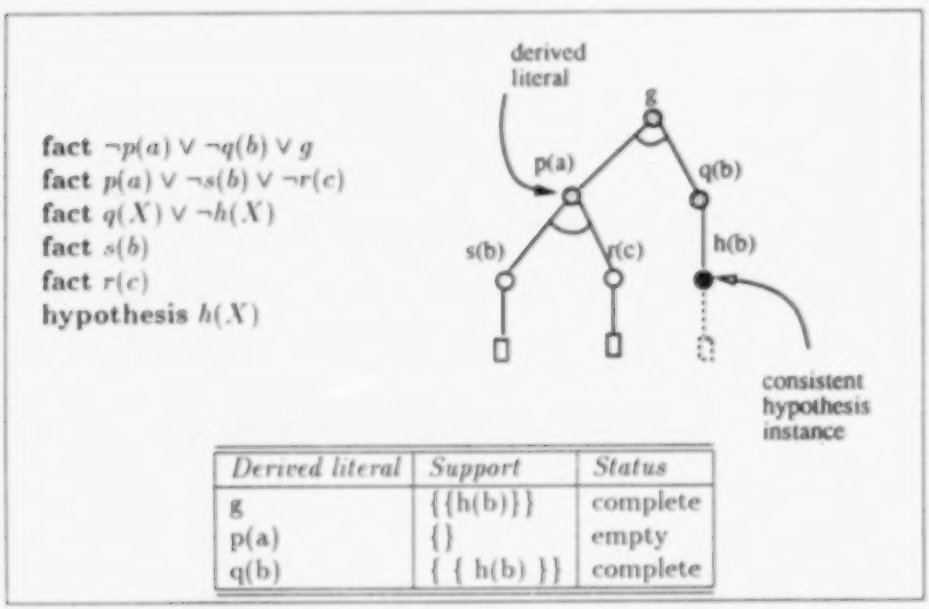
[5]One of the criticisms of TMSs is the amount of space they require to store the derived information. Our incremental strategy may turn out to be better than others, but still size of the tables might grow rapidly. One possible solution to address this problem is to use the standard memory management techniques, e.g., throwing out least used information to maintain the desired size of the tables. However, further investigations are required to maintain the logical meaningfulness of these tables.

Here, we would like to completely avoid redundancy by assuming that space is not the problem.

Derived literals

While constructing possible explanations for a given query, a number of subqueries have to be answered—subgoals to be proved. The proofs of the subgoals may or may not require hypothesis instances. To explain a query $\mathcal{Q}$, a proof tree $F \cup \mathcal{E} \cup \{\mathcal{Q}\}$ is constructed, where $\mathcal{E}$ is a set of hypothesis instances drawn from H and $\mathcal{E}$ is consistent. This proof tree is called as an *explanation tree.*

Definition 4 (Derived literal) *Let $F \cup \mathcal{E} \cup \{\mathcal{Q}\}$ be an explanation tree. Then each atomic non-leaf node n in that explanation tree is a derived literal if there is no literal, say l, in the subtree rooted at n such that $\neg l$ is an ancestor of n. I.e., $F \cup \mathcal{E}' \vdash n$, where $\mathcal{E}' \subseteq \mathcal{E}$.*



Derived literal	*Support*	*Status*
g	{{h(b)}}	complete
p(a)	{}	empty
q(b)	{ { h(b) }}	complete

Figure 10: An explanation tree and derived literals

The derived literals, when recorded as explicit beliefs in the tabular form, need not be re-derived if required in the future computations. A simple example is illustrated in Figure 10. An explanation tree for goal g is constructed

from the given facts and hypotheses, i.e., $F \cup \{h(b)\} \models g$. The nodes g, $p(a)$ and $q(b)$ are derived literals in the explanation tree. Each of these literals is recorded as a tuple of the form $< literal, support, status >$ (see Table in Figure 10), where *literal* is a derived literal, *support* is a set of explanations for that *literal*, and *status* indicates whether *support* is empty, partial or complete. If *support* is empty then the goal can be derived without using hypotheses. Partial *support* means each element of support forms an explanation for the *literal*, but it may not include all possible explanations. A complete status indicates that the *support* contains all possible explanations for *literal*.

The following theorems give some properties of the truth maintenance table containing the derived literals:

Theorem 5 *Let* $< \mathcal{L}, \mathcal{S}, partial/complete >$ *be a derivation table tuple. Suppose* $\mathcal{L}$ *is a query to be answered. Then each* $S \in \mathcal{S}$ *is an explanation for* $\mathcal{L}$.

Theorem 6 *If* $< \mathcal{L}, \mathcal{S}, empty >$ *is a tuple, then* $\mathcal{S}$ *is empty and* $F \models \mathcal{L}$.

Theorem 7 *Let* $< \mathcal{L}, \mathcal{S}, complete >$ *be a tuple. Then*

- *A set of hypothesis instances* S *is an explanation for a query* $\mathcal{L}$ *iff* $S \in \mathcal{S}$.
- *The set* $\mathcal{S}$ *corresponds to an ATMS label.*

The proofs of the above theorems are straight forward from the definition (see Def 4) of the derived literals.

Potential crucial literals & Nogoods

The specification of basic Theorist suggests that the consistency of a hypothesis instance can be verified by failing in all attempts to find a proof of its negation. During that check for consistency, we are not allowed to assume a hypothesis instance which is not already an element of a partially constructed explanation. We extend the computation to consider combinations of inconsistent hypothesis instances using the notion of *consistency tree* (see Def. 3).

Let us consider an example (as shown in Figure 11) to illustrates how a consistency tree can be constructed. Figure 11(c) is a consistency tree for an hypothesis instance $\neg h1(a)$ with respect to the Theorist database. In this example, to explain g, the *proof generator* has to prove the two subgoals $p1(a)$ and $p2(a)$. If $p1(a)$ is selected first, its proof would require hypothesis instance $h1(a)$. Before using $h1(a)$ in the explanation tree, the *consistency checker* is requested to test the consistency by establishing unprovability of $\neg h1(a)$. Figure 11(a) shows one of the finitely failed branches. However, if we use new hypothesis instances to extend the failure branches, some of them become success branches—consistency trees. Figure 11(b) shows a consistency tree as stated in Def. 3. The proof tree in Figure 11(c), where each node is an atomic subgoal, is logically equivalent to the tree shown in Figure 11(b).

Since the consistency test is the most essential and computationally the most expensive, recording consistency trees in the form of truth maintenance tables will significantly improve efficiency of the underlying problem solving system. Here, we describe two kinds of truth maintenance tables for recording information from consistency trees: potential crucial literals and nogoods.

Definition 8 (Potential crucial literal) *An atomic internal node n in a consistency tree is a potential crucial literal if there is no literal, say l, in the subtree rooted at n such that $\neg l$ is an ancestor of n.*

The table of potential crucial literals (see Figure 12) consists of 3-place tuples:

$$< \textit{potential crucial literal}, \textit{inset}, \textit{outset} >.$$

A potential crucial literal with non-empty *inset* may be used to discriminate between theories containing the *inset* and the *outset* respectively. These literals are simply called *crucial literals* [153]. For instance, the truth value of the literal $p2(a)$ discriminates theories containing $\{h3(a)\}$ and $\{h1(a), h2(a)\}$. The literal $f(a)$, whose *inset* is empty, is a derived literal. The following are the definitions of *inset* and *outset*:

Definition 9 (inset) *Let T be the set of hypothesis instances in a consistency tree tree($\neg h_i$), i.e., $F \cup T \vdash \neg h_i$. Let pcl be a potential crucial literal in the tree tree($\neg h_i$). The set T' of hypothesis instances contained in the subtree rooted at pcl is called an inset w.r.t. pcl.*

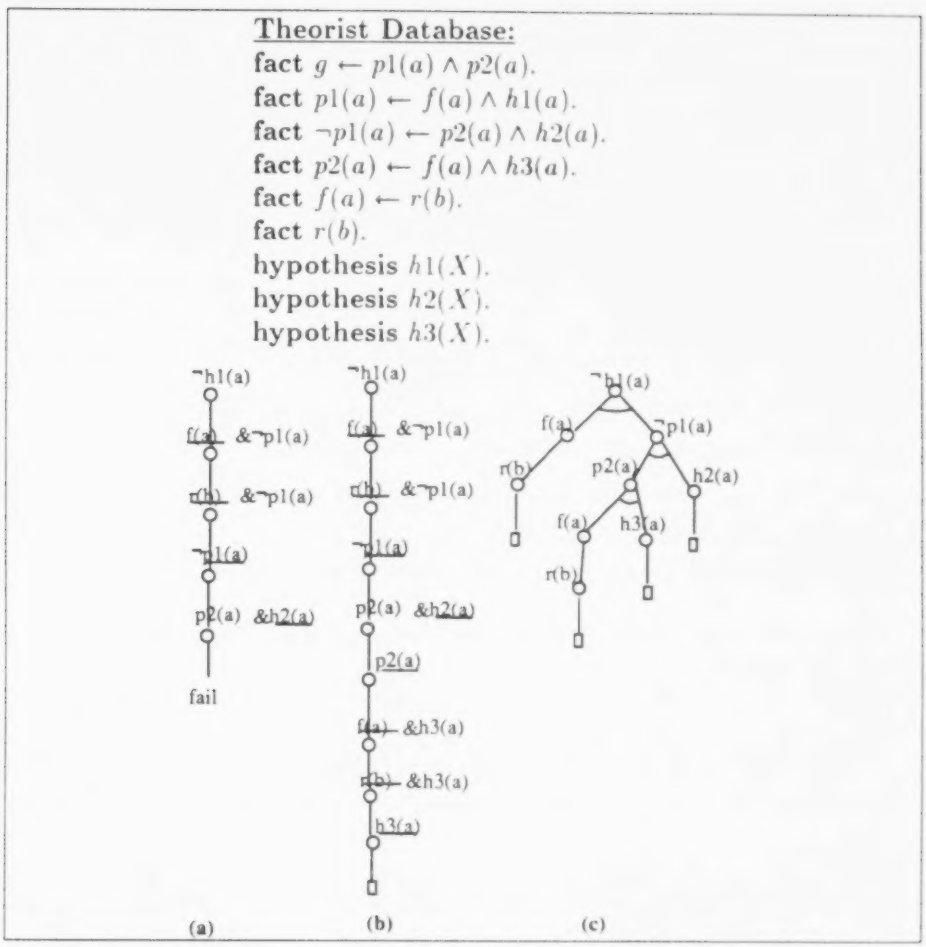
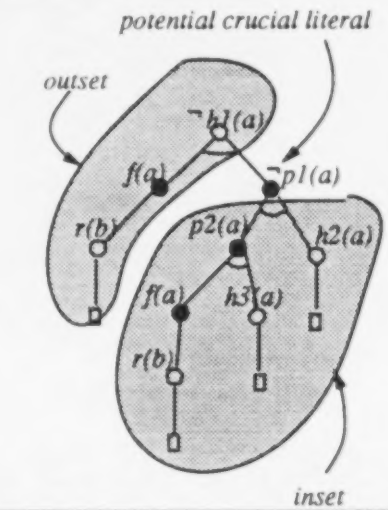


Figure 11: Extending failure tree to consistency tree

Definition 10 (outset) *Let T be the set of hypothesis instances in a consistency tree tree($\neg h_i$), i.e., $F \cup T \vdash \neg h_i$. Let pcl be a potential crucial literal in the tree tree($\neg h_i$). Let T' be the set of hypothesis instances contained in the subtree rooted at pcl. Then the set $T \cup \{h_i\} - T'$ is called an outset w.r.t. to pcl.*



potential crucial literal	*inset*	*outset*
$\neg p1(a)$	$\{h3(a), h2(a)\}$	$\{h1(a)\}$
$p2(a)$	$\{h3(a)\}$	$\{h1(a), h2(a)\}$
$f(a)$	$\{\}$	$\{h1(a), h2(a), h3(a)\}$

Figure 12: Identifying potential crucial literals, insets and outsets

Figure 12 indicates how a potential crucial literal and its *inset* and *outset* can be identified in a consistency tree. For a potential crucial literal, $\neg p1(a)$, a set of hypothesis instances in the subtree rooted at that literal forms its *inset*, $\{h3(a), h2(a)\}$. A set of hypothesis instances in the rest of tree forms *outset*, $\{h1(a)\}$, for that literal. A tuple $< pcl, inset, outset >$ indicates that *inset* supports *pcl* and *outset* supports $\neg pcl$.

Lemma 11 *Let $< pcl, inset, outset >$ be a tuple and F be a set of facts. Then inset is an explanation for pcl (i.e., $F \models inset \rightarrow pcl$) iff $F \cup inset$ is consistent.*

Lemma 12 *Let $< pcl, inset, outset >$ be a tuple and F be a set of facts. Then outset is an explanation for $\neg pcl$ (i.e., $F \models outset \rightarrow \neg pcl$) iff $F \cup outset$ is consistent.*

From each tuple in the table, we can implicitly derive a new tuple such that one is the dual of the other.

Lemma 13 *Let $< pcl, inset, outset >$ be a tuple. Then we can derive its dual $< pcl', inset', outset' >$ such that $pcl' = \neg pcl$, $inset' = outset$, and $outset' = inset$.*

Figure 11(c) shows that $Facts \models (h3(a) \wedge h2(a)) \rightarrow \neg h1(a)$, i.e., $Facts \cup h1(a) \wedge h2(a) \wedge h3(a) \vdash false$. So, the set $\{h1(a), h2(a), h3(a)\}$ is inconsistent or nogood.

Definition 14 (nogood) *Suppose t is an hypothesis instance ($t \in H$), F is a set of facts, and $T \subseteq H$. If there is a consistency tree for t such that $F \cup T \vdash \neg t$, then $T \cup \{t\}$ is a* **nogood**.

A *nogood*, N, is a *minimal nogood* if there is no *nogood* N' such that $N' \subset N$.

Theorem 15 *Let $< pcl, inset, outset >$ be a tuple and F be a set of facts. Then $F \cup inset \cup outset \vdash false$, i.e., $inset \cup outset$ is a nogood.*

Proof: Follows from the definition of nogoods (see Def. 14). ■

The following theorem states that the potential crucial literals with empty *inset* set are theorems established from the facts, and need not be re-derived.

Theorem 16 *Suppose pcl is a potential crucial literal such that the respective $inset = \{\}$. Then, $Facts \models pcl$ (i.e., pcl is derived implicit belief).*

Proof: From definition of the consistency tree and the inset, it is clear that a potential crucial literal, say *pcl*, is a consequence of facts and the hypothesis instances in the corresponding *inset*. That is, $Facts \cup inset \models pcl$. Therefore, if $inset = \{\}$, then $Facts \models pcl$. ■

4.3.4 Incremental Explanation Construction

Having constructed an explanation, existing versions of Theorist discard all record of how each subgoal of an observation sentence was proved—which instances of hypotheses were inconsistent and which subgoals were proved without using hypotheses. Each subsequent derivation researches the search space.

Our alternative is to extend the basic Theorist framework with truth maintenance tables to record the useful information computed in each derivation, then use this information to efficiently guide subsequent search. As this information monotonically increases, the search space is reduced. Here, we describe an incremental proof strategy for consistency checking and explanation construction. The new proof procedures for explanation construction take advantage of information contained in the truth maintenance tables.

Definition 17 (New consistency checker) *Let F be a set of facts; H be a set of possible hypotheses; and q be a query to be explained. Suppose T is a set of hypotheses instances drawn from H such that $F \cup T \vdash q$. Then*

- *If there exists a $t \in T$ and $T' \subseteq T$ such that $F \cup T' \vdash \neg t$, then $\forall T'$, $T' \cup \{t\}$ is recorded as a nogood.*
- *If for all $t \in T$, $F \cup T - \{t\} \not\vdash \neg t$, i.e., each T is an explanation for q, then construct consistency trees for each t and record all possible potential crucial literals from that tree.*
- *The procedure for constructing a consistency tree (see Def. 3) is extended by the following steps, whihc enables the consistency checker to use truth maintenance tables:*
  - *If A is a unit goal and A is unified with some derived literal, say l, in a tuple $< l, \{\}, empty >$, then A has no descendant (it is a consequence of the set of facts).*
  - *If A is a unit goal and A is unified with some potential crucial literal, pcl, in a tuple $< pcl, \{\}, outset >$, then A has no descendant (it is a consequence of the set of facts).*
  - *If A is a unit goal and A is unified with some derived literal, l, in a tuple $< l, S, complete >$, then each $s \in S$ is a support set for A (we don't have to explore further descendents).*

- *If A is a unit goal and A is unified with some pcl, in a tuple $< pcl, inset, outset >$ such that $inset \neq \{\}$), then all such inset's are support sets for A (we don't have to explore further descendents for other support sets).*

It is important to note that, in the above procedure, if the goal A unifies with some *pcl*, then all derivable support sets are available in the table of potential crucial literals. Similarly, if the goal A unifies with some derived literal l in a tuple $< l, S, complete >$, then the set S contains all possible explanations for l. Consequently, the behavior of the above proof procedure is improved by avoiding re-computing previously derived conclusions from the facts, *and* tentative conclusions derived from the facts and hypotheses instances.

However, we must take extra care when a subgoal is not a consequence of facts alone but requires some hypothesis instances. In the consistency tree, we only need to record the set of hypothesis instances required to derive the subgoal. But, when constructing an explanation, if a subgoal is unified with some literal from either the derived literal table or the potential crucial literals table such that the *support set* or *inset* are not empty, then we must check consistency of the elements of *inset* while adding to the partial explanation. When the *inset* is added to the partial explanation, the *outset* becomes nogood. This nogood can not be treated as a global nogood, because the set of hypothesis instances (i.e., partial explanation) under consideration may turn out to be inconsistent or *outset* may be consistent for another explanation. So, we characterize such a nogood as a *local nogood*. This *outset* remains inconsistent only for the explanation being constructed. Similarly, we have to check consistency of the *support sets* used from the table of derived literals. Although *support sets* are already shown to be consistent independently, their combination with the partially constructed explanation may cause inconsistencies.

Definition 18 (local nogood) *Suppose, while constructing an explanation, a subgoal unifies with some potential crucial literal, say pcl, in a tuple $< pcl, inset, outset >$, and the inset is verified as a consistent partial explanation. Then the outset and its supersets must not be considered for the explanation being constructed. The outset becomes inconsistent, and is referred to as a* local nogood.

We give an incremental proof procedure for constructing an explanation for a given set of observations (i.e., a query). The proof generator consults the truth maintenance tables each time when it needs to prove some goal. Here, we the notation T_{pcl} (the table of potential crucial literals), T_{dl} (the table of derived literals), NG (the table of minimal nogoods), NG_{local} (the table of local nogoods) and $E_{partial}$ (partial explanation).

Definition 19 (New proof generator) *Let Q be a query, F a set of facts, and H a set of hypothesis instances. Constructing explanations for Q requires determination of all success branches of the AND/OR derivation tree for $F \cup H \cup \{Q\}$. Given the tables T_{pcl}, T_{dl}, NG, and NG_{local}, an incremental proof procedure for constructing each success branch of this tree is defined as follows:*

1. *Initialize $NG_{local} := \{\}$ and $E_{Partial} := \{\}$.*
2. *The root node Q, is a goal.*
3. *A unit goal A is a leaf node if A unifies with some element of the table T_{pcl}, say pcl in a tuple $< pcl, \{\}, outset >$.*
4. *A unit goal A is a leaf node if A unifies with l in tuple of the table T_{dl}, say $< l, S, empty >$.*
5. *A unit goal A is a leaf node if A unifies with some element of the table T_{pcl}, say pcl of a tuple $< pcl, inset, outset >$ such that $inset \neq \{\}$, and $inset$ is consistent with $E_{partial}$.[6] Then, $E_{Partial} := E_{Partial} \cup inset$ and $NG_{local} := NG_{local} \cup outset$.*
6. *A unit goal A is a leaf node if A unifies with some element of the table T_{dl}, say l of a tuple $< l, S, complete >$[7] such that $S \neq \{\}$ and some element of $s \in S$ is consistent to add with $E_{partial}$, similar to step 5. Then, $E_{Partial} := E_{Partial} \cup s$.*
7. *If a node is a conjunction of subgoals, say $\leftarrow A_1, ..., A_i, ..., A_n$, then find a rule $A \leftarrow B_1, ..., B_m$ such that A_i and A are unifiable, then the successor node is*

[6] Here, consistency means for all $t_i \in inset$, $F \cup E_{Partial} \cup \{inset - t_i\} \not\vdash \neg t_i$, and there is no $ng \in NG \cup NG_{local}$ such that $ng \subseteq E_{Partial}$.

[7] If we are looking for only one explanation then we can also consider the tuples with *partial* status.

$$(A_1, ..., A_{i-1}, B_1, ..., B_m, A_{i+1}, ..., A_n)\theta,$$

where θ is a most general unifier of A_i and A.

8. *If A is a goal and $\neg A$ is an ancestor of A, then A has no descendant (by reductio ad absurdum).*
9. *If A is a goal which unifies with an instance of hypothesis $h_i \in H$ and h_i is consistent, then A has no descendant. Update the partial explanation: $E_{Partial} := E_{Partial} \cup \{h_i\}$. (Note that this is where consistency checker will be invoked.)*
10. *Nodes which are empty goals have no descendants.*
11. *When a proof tree is found, the explanation is $T_{Partial}$.*

To illustrate the incremental proof mechanism as described above, we consider the example shown in Figure 11. We want to construct explanations for the observation g. We are given $NG = \{\}$,$T_{dl} = \{\}$ and $T_{pcl} = \{\}$. The following traces the steps of explanation construction:

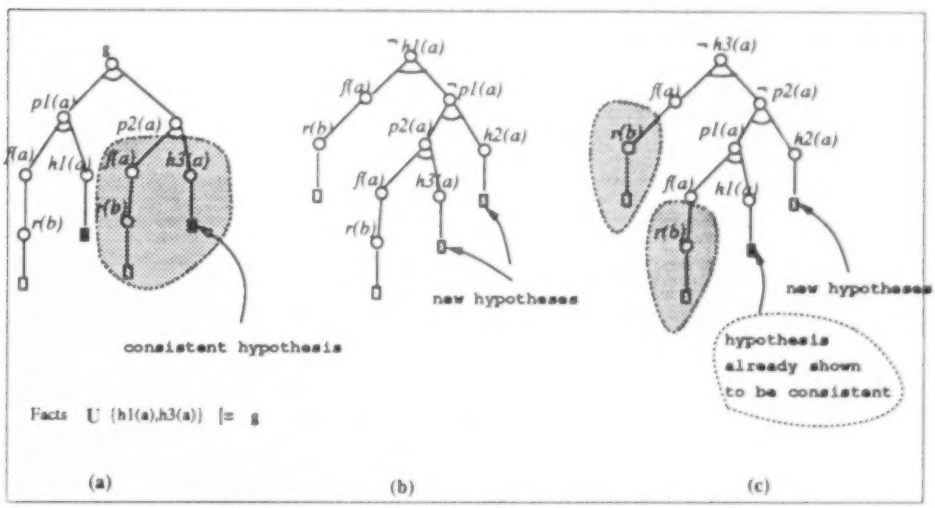


Figure 13: Incremental explanation construction for goal g

(a) $NG_{local} := \{\}$, $E_{Partial} := \{\}$.

(b) The root g has two descendants: $p1(a)$ and $p2(a)$.

(c) Suppose $p1(a)$ is selected first. It has two subgoals: $f(a)$ and $h1(a)$. Further, $f(a)$ has descendant $r(b)$ which is a leaf, and $h1(a)$ is an hypothesis instance which is subject to a consistency test.

(d) Determine all consistency trees for $h1(a)$. There is only one consistency tree for $h1(a)$ as shown in Figure 13(b). The consistency tree gives information:

$$NG = \{\ \{h1(a), h2(a), h3(a)\}\ \}$$

$\mathbf{T_{pcl}} =$

potential crucial literal	*inset*	*outset*
$\neg p1(a)$	$\{h3(a), h2(a)\}$	$\{h1(a)\}$
$p2(a)$	$\{h3(a)\}$	$\{h1(a), h2(a)\}$
$f(a)$	$\{\}$	$\{h1(a), h2(a), h3(a)\}$

(e) $E_{partial} := \{h1(a)\}$.

(f) Now the procedure has to prove the subgoal $p2(a)$. Since $p2(a)$ unifies with an element in T_{pcl}, it has no descendant and the inset $\{h3(a)\}$ may be added to $T_{partial}$ if $T_{partial} \cup \{h3(a)\}$ is consistent. Figure 13(c) shows the consistency tree for $h3(a)$, which is constructed using the truth maintenance tables:

- $\neg h3(a)$ has two descendants $f(a)$ and $\neg p2(a)$.
- $f(a)$ has no descendant, since it is an established theorem in T_{pcl}.
- To prove the subgoal $\neg p2(a)$, the prover needs hypothesis instances $h1(a)$ and $h2(a)$. However, it can only use $h1$, i.e., $h3(a)$ is consistent. The consistency checker extends that failure branch using hypothesis instance $h2(a)$ to construct constency tree.
- Update T_{pcl} and NG. The updated T_{PCL}[8] is as follows:

[8] In this table, rows 1 & 2, and rows 3 & 4 could be merged into one as one is the dual of the other.

$\mathbf{T_{pcl}}$ =

potential crucial literal	*inset*	*outset*
$\neg p1(a)$	$\{h3(a), h2(a)\}$	$\{h1(a)\}$
$p1(a)$	$\{h1(a)\}$	$\{h2(a), h3(a)\}$
$p2(a)$	$\{h3(a)\}$	$\{h1(a), h2(a)\}$
$\neg p2(a)$	$\{h1(a), h2(a)\}$	$\{h3(a)\}$
$f(a)$	$\{\}$	$\{h1(a), h2(a), h3(a)\}$

Set $NG_{local} := NG_{local} \cup \{h2(a), h1(a)\}$. Note that the shadowed areas in Figure 13(a) and (c) are not searched, as these derivations already exist in the truth maintenance tables.

(g) $T_{partial} := \{h1(a)\} \cup \{h3(a)\}$.

(h) No subgoals remain; thus $Explanation := \{h1(a), h3(a)\}$.

(i) Finally, useful information from the explanation tree for g is extracted. Here, we should only record the information which is not in T_{PCL}, e.g., derived literals $f(a)$, $p1(a)$, $p2(a)$ already recorded in T_{PCL}. In this case, we will have only one entry as given below:

$$T_{dl} = \{< g, \{h1(a), h3(a)\}, complete >\}$$

4.3.5 Updating the Truth Maintenance Tables

In the previous section, we have described incremental procedures for generating ordinary proofs and testing consistency of hypothesis instances. The search efficiency of these procedures depends on the truth maintenance tables. The truth maintenance tables are dynamically changing with new information received from the incremental proof procedures. Thus, the truth maintenance component must update the tables each time it receives new information. Here, we specify algorithms to dynamically maintain the truth tables: T_{pcl}, NG, and T_{dl}.

Algorithm 20 *The following steps are used to incrementally compute and maintain derived literals, nogoods and potential crucial literals:*

1. *For any hypothesis instance whose consistency is being checked, say h_i, construct all success branches of the tree $F \cup H \cup \{\neg h_i\}$ using the definition of consistency tree.*
2. *For each success branch of the derivation tree, $F \cup T \vdash \neg h_i$, where $T \subset H$, do the following:*
   - *Add the set $\{T \cup h_i\}$ to NG, using Algorithm 21.*
   - *Record all potential crucial literals using Algorithm 22.*
3. *For each explanation tree for a goal g, $F \cup E \models g$, record the derived literals using Algorithm 23.*

Algorithm 21 (Updating NG) *: Suppose ng is a nogood to be added to NG. Then*

- *if $NG = \{\}$, then add ng to NG;*
- *if $ng_1, ..., ng_i \in NG$ such that $ng \subset ng_1, ..., ng \subset ng_i$, then add ng to set NG and remove all supersets $(ng_1, ..., ng_i)$ from NG;*
- *if there is an element $ng_i \in NG$ such that $ng_i \subseteq ng$, then ignore ng;*
- *if none of the above is true, then add ng to NG.*

The input to the above algorithm is a *nogood* derived from a consistency tree. The algorithm then ensures that the information about the nogood is not lost; and the sets of nogoods are minimal w.r.t. set inclusion.[9]

Algorithm 22 (Updating T_{pcl}) *Let $pcl_1, ..., pcl_n$ be the potential crucial literals in the consistency tree. Suppose for each pcl_i, In_i and Out_i are the inset and outset. Then the following steps are used to update the table for each tuple $< pcl_i, In_i, Out_i >$:*

1. *If there is no entry in the table, then create new table entry for the tuple $< pcl_i, In_i, Out_i >$.*

[9] The minimality is considered with respect to the currently compiled knowledge. However, it may be altered by future changes.

2. *If there exists a tuple* $< pcl_i', In_i', Out_i' >$ *in the table such that* $pcl_i \equiv pcl_i'$ [10], $In_i' \subseteq In_i$ *and* $Out_i' \subseteq Out_i$, *then replace the old tuple by new one.*

3. *If there exists a tuple* $< pcl_i', In_i', Out_i' >$ *in the table such that* $pcl_i = \neg pcl_i'$, $In_i' = Out_i$ *and* $Out_i' = In_i$, *then the new tuple is the dual of an existing tuple. Thus, it is redundant to add this tuple to the* T_{PCL}.

4. *If conditions 1, 2 and 3 are not true, then create a new table entry to record the tuple* $< pcl_i, In_i, Out_i >$.

The input to Algorithm 22 is a set of tuples of potential crucial literals and their respective *insets* and *outsets*. Step 1 creates a new table entry if the table is empty. Since there may be more than one set of *insets* and *outsets* for a potential crucial literal, step 2 ensures that the *inset* and the *outset* are minimal w.r.t. set inclusion. Step 3 ignores the tuple if it is a dual of the existing one. Finally, we create a new table entry if required (i.e., conditions in 1,2 and 3 are not true.).

Algorithm 23 (Updating T_{dl}) *While generating explanations for a goal g, there are two situations: looking for only one explanation* E_i, $F \cup E_i \models g$; *or finding all possible explanations* $E_1, ..., E_n$, *i.e.,* $F \cup \{E_1, ..., E_n\} \models g$. *The following steps are used to update the table* T_{dl}:

1. $F \cup E_i \models g$, *and only one explanation tree is available and the derived literals in that tree are* $l_1, ..., l_k$, *and for each* l_i *the corresponding tuple is* $< l_i, s_i, status_i >$, *where* s_i *is a support set and* $status_i$ *is the status. Then the update operations for each* l_i *are as follows:*

   (a) *If there is no entry in the table, then create new table entry* $< l_i, s_i, partial >$. *If* s_i *is empty, then this tuple should be* $< l_i, \{\}, empty >$.

   (b) *If there exists a tuple* $< l_i', S', partial >$ *such that* $l_i \equiv l_i'$ *and there is some element* $s \in S'$.

   - *If* $s_i \subset s$, *then replace* s *by* s_i.
   - *If* $s_i \supset s$, *then ignore it.*
   - *Otherwise,* $S' := S' \cup \{s_i\}$.

[10] Note that all pcl_i are literals, so '$\equiv$' is a simple table retrieval and unification operation.

2. *$F \cup \{E_1, ..., E_n\} \models g$, i.e., all possible explanation trees are computed for the goal g. For each explanation tree, the derived literals are $l_1, ..., l_k$.*
   (a) *Each literal l_i is recorded in the form of a tuple $< l_i, S_i, complete >$, if there is no set of hypothesis instances, say s' such that $s' \notin S_i$. Otherwise, the literal l_i is recorded as $< l_i, s_i, partial >$.*
   (b) *If there exists a tuple $< l'_i, S', partial >$ such that $l_i \equiv l'_i$, and conditions of the previous step satisfy then replace that tuple by $< l_i, S_i, complete >$.*

The input to the above algorithm is a set of tuples of derived literals. The table of derived literals has to be updated with respect to this input in such a way that no previously derived information is lost and the second argument of each tuple contains a set of minimal supports.

4.3.6 Empirical Efficiency

While discussing empirical efficiency of the extended-Theorist, the following aspects can be considered:

- The table of potential crucial literals allows the reasoner to avoid redundant computation of derived implicit beliefs and of many potential partial/full explanations.
- The table of derived literals also helps to avoid redundant derivations.
- The table of nogoods supports the dynamic pruning of the search space without actually exploring that space.
- A subset of potential crucial literals may be used to incrementally discriminate between incompatible explanations. That helps reduce the possibly exponentially large explanation space (i.e., solutions for the problem).
- In the view of new knowledge, the truth maintenance tables can be efficiently updated.
- The incremental consistency check does not avoid backtracking completely, our search procedure can be extended to learn from backtracking as well [157]. The idea is that the reasons for dead-ends can be

recorded unextendable partial explanations to avoid discovery of the same backtrack point. This idea is very useful in solving constraint satisfaction problems. (See Chapter 5).

4.4 Relationship with Other TMSs

To compare our approach, we first summarize advantages and limitations of the TMS and the ATMS, then we discuss how our framework captures the advantages and overcomes many of the limitations.

The TMS has the following main objectives:

- to avoid redundant computation of inferences by employing a caching mechanism;
- to support nonmonotonic reasoning (i.e., new facts may invalidate old conclusions); and
- to maintain consistency of dynamically changing knowledge bases by using dependency-directed backtracking to remove the inconsistencies, and by recording inconsistencies as *nogoods* to avoid repetition of the same contradiction.

De Kleer [25, 26] has pointed out many limitations of the TMS; here are the main ones:

- the TMS only allows one solution to be considered at a time; that means we can not compare multiple plausible solutions (possibly mutually contradictory);
- dependency directed backtracking improves chronological backtracking, but in an *a posteriori* way (i.e., after having discovered a contradiction).

Our approach provides a simple and elegant approach for truth maintenance without the limitations of the TMS. It does not require global consistency (i.e., mutually contradictory solutions may co-exist) and reduces search space in an *a priori* way by incrementally computing nogoods. Similar to the TMS, our system supports nonmonotonic reasoning.[11] Since consistency checking

[11] Its relationship with other formalisms of nonmonotonic reasoning is discussed in [134].

is the most expensive (i.e., semi-decidable) and most essential for hypothesis-based reasoning, the best way to improve the search efficiency is to extract useful information from each consistency tree and use it to make future computations more efficient.

The ATMS is based on the similar ideas, but primarily aimed at overcoming the limitations of TMS by manipulating assumptions rather than justifications. The ATMS has several advantages over the TMS: no backtracking is required; simultaneous consideration of multiple contexts which may be mutually contradictory; simplify the complex data structure of the TMS etc. However, there are the following main problems with the ATMS:

- Reiter and de Kleer [147] have shown that the ATMS (a propositional theorem prover) can compute all solutions (explanations) when the goal (observation) is a single literal. However, finding one of them may require exponential time [164]. Therefore, even after such restrictions on the language the ATMS can not guarantee computational tractability.

- It does not provide any mechanism to incrementally discriminate between competing solutions. The only way to do so is to first compute all solutions, then apply some selection criteria. That is at best very expensive, particularly in the diagnostic reasoning problems where the number of possible diagnoses may be exponentially large.

- The ATMS has to compute all of the nogoods in order to completely avoid backtracking (i.e., full lookahead search [67]). However, it has been shown, both experimentally and analytically, that full lookahead or partial lookahead is inferior to the forward checking [67].

- Unlike the TMS, the basic ATMS [26] does not support nonmonotonic reasoning and is not suitable for the problems requiring only one solution.

Our approach is committed to retain the expressive power of full clausal logic. Consequently, problems caused by the weakness of the propositional logic do not arise. In addition, our framework of hypothetical reasoning does not compute all solutions if not required; nogoods are incrementally computed using consistency trees, which has been shown to be equivalent to *forward checking* (see Chapter 5); and crucial literals may be used to incrementally

distinguish incompatible competing solutions, which can reduce the solution space from exponential to linear (see Chapter 7). However, similar to forward checking, the consistency trees do not avoid backtracking completely. We have shown in [157] that the integration of forward checking using consistency trees and truth maintenance tables gives much better results. For instance, in the four queens problem the best heuristics used with forward checking requires 2 backtrack and 34 consistency test [181] where as our approach needs 2 backtracks and 26 consistency trees. Of course, for multiple solutions our approach performs better.

In comparison to Logic-based Truth Maintenance Systems (e.g., LTMS [109]), we provide a framework which is expressively rich and nonmonotonic.

4.5 Summary

We have presented an extension to the Theorist hypothetical reasoning system [136, 134], by introducing an incremental proof strategy for consistency checking and explanation construction. The extension preserves the original simplicity and elegance of Theorist, and addresses a major problem of consistency-based nonmonotonic reasoning—computational efficiency. In the extended system, the burden of computation is shifted to the problem of retrieval and updating the table of potential crucial literals. We have elsewhere [153] shown that crucial literals may be used to perform experiments or tests to incrementally discriminate theories. The relationship with other TMSs suggests that our approach not merely provides an adequate representation system but also captures advantages of the traditional truth maintenance systems while overcoming many of their problems.

Chapter 5

Constraint Satisfaction in Hypothetical Reasoning

5.1 Introduction

The constraint satisfaction paradigm has been used as a tool for efficient problem solving and investigated with great interest, particularly in AI, for the last two decades. The word *constraint* is a general term that could be defined as a relation over a Cartesian product, a Boolean predicate, a fuzzy relation, an algebraic equation, an inequality, a Horn clause in a logic program and various arbitrary complex symbolic relationships [102]. A *constraint satisfaction problem* (CSP) is to determine a consistent assignment of values for a set of variables satisfying the specified constraints. Let I be a set of finite variables $\{X_1, X_2, ..., X_n\}$; variables may be instantiated using values from their respective domains $D_1, D_2, ..., D_n$; and a set of constraints $c(X_{i_1}, ..., X_{i_k})$, between k variables from I, which is a subset of the Cartesian product $D_{i_1} \times ... \times D_{i_k}$ that specifies which values of the variables are compatible. Then a set of values that satisfy all the constraints would form one solution to the CSP.[1] However, the constraint solver may be required to find either one or all solutions for a given CSP.

The notion of constraint satisfaction in AI exploits the idea that our knowledge about the real world can be easily expressed in terms of what is

[1] The general CSP is NP-complete [123].

allowed and what is not allowed; and such a formulation of knowledge may be used by an intelligent agent to prune unnecessary search space without actually exploring it. To do so, an intelligent constraint solver has to cope with the two fundamental problems: precise specification of constraints, and the use of that specification to efficiently search for one or all solutions for a given CSP.

Logic has been considered to be an appropriate language to specify constraints (e.g., [148]). However, most of the logic-based constraint solving systems impose some restrictions on the logical language (e.g. propositional logic in TMS based approaches [37, 26, 109], Horn clause constraint logic programming systems [180, 81, 18], etc.). Consequently, many constraints are difficult to represent. E.g., "every fungus is a mushroom or a toadstool" or "every thing which is neither a mushroom nor a toadstool is not a fungus" can only be represented in a non-Horn clause (i.e., $mushroom(X) \vee toadstool(X) \leftarrow fungus(X)$) [88].

Over the last two decades, a number of techniques have been evolved for constraint solving. These techniques may be characterized into three classes [102]: *backtracking, consistency algorithms* and *cooperative algorithms*. Backtracking and consistency-based approaches are most commonly known, where as cooperative algorithms are used in complex problem domain where thousands of processors can work cooperatively (e.g., computer vision [23]).

The standard backtracking (e.g., generate and test or chronological backtracking) suffers from well known *thrashing* behavior and can be very inefficient for solving CSPs [101, 45, 67]. Many improvements have been proposed such as *intelligent backtracking, selective backtracking, dependency directed backtracking and backmarking* (e.g., [174, 21, 107, 54]). However, the improved versions of backtracking are quite conservative in the way they reduce the search space, that is, in *a posteriori* way after having discovered a failure or dead-end. Sometimes the reasons for dead-ends are recorded in the form of new constraints as conflicting sets and are used for improving the future backtracking [31]. Similar ideas have been used in STRIPS with MACROPS [43] and in a general problem solver [89], as well.

The consistency-based approaches seem to have begun with Waltz's algorithm [183] and Montanari's work on applying constraints to picture processing [123]. Later, Mackworth [101] presented a similar idea in a more formal way. According to him, the consistency-based techniques can be better understood when we think of them as removing local inconsistencies,

which may cause inconsistencies in the neighboring arcs. These are in turn removed so that the constraint propagation eventually reaches a fixed point consistent assignment of a value for each variable. He defines consistency of a network relation in terms of *node consistency*, *arc-consistency* and *path-consistency*. Freuder [45] then generalizes these different forms of consistency as *k-consistency* where $k \leq n$, n is number of nodes in a network. Further, he shows that sufficient preprocessing can eliminate most or all futile backtracking [46]. Sometimes consistency-based constraint solving is referred to as forward reasoning which includes the heuristics such as *forward checking*, *partial lookahead* and *full lookahead* [67]. Each of these heuristics has some trade-off between the cost of pre-processing (i.e., consistency checks) and backtracking. An experimental and analytical study [67] shows that the forward checking performs better than the others which requires less consistency checks and relatively more backtracking than the other two heuristics.

Since the use of full clausal logic in our hypothetico-deductive reasoning system makes the problem of constraint satisfaction undecidable, the traditional complexity measures become void for analyzing computational behaviour. What remains is some empirical method or qualitative argument to analyze the performance of constraint solving. However, there are the following good reasons to cast the problem of constraint satisfaction in hypothetical reasoning:

- Our knowledge about constraints (i.e., what is allowed or what is not allowed) and finite domains of variables is a kind of meta-knowledge; the hypothetical reasoning framework supports a logical language of *facts* and *hypotheses* that provides the user a greater flexibility and expressive power to represent such knowledge at the meta-level. Representing domain knowledge at the object-level (e.g., introducing finite domains in logic programming [179, 180, 181]) may be problematic from the user's viewpoint.

- Most logic-based constraint solving systems are more or less restricted, where as our hypothetical reasoning system allows any arbitrarily complex constraints that can be expressed in first-order logic.

- While solving constraints, sometimes we want to deal with nonmonotonicity, for example, the problem of diagnosis [146]. Hypothetical

reasoning captures nonmonotonicity by maintaining solutions of CSPs as potentially refutable scientific theories.

The motivation for using a restricted logical language, e.g., propositional calculus, in constraint solving systems (e.g., TMSs [37, 109, 26]) is to have a decidable system. However, decidability does not guarantee computational tractability. We prefer full expressive power, and rely on heuristics for empirical efficiency. Efficiency of the hypothetical reasoning-based constraint solver depends on unifying consistency-based approach with CSPs (e.g., [101, 67]), truth maintenance (e.g., [26]), learning in backtracking to avoid re-discovery of the same backtrack points [31], and an incremental mechanism to discriminate between incompatible solutions using crucial literals—the user guidance (see chapter 7).

This chapter presents a hypothetico-deductive reasoning-based constraint solving system[2] which provides an adequate expressive capability to specify arbitrarily complex constraints and an empirically efficient search procedure. Basically, the search efficiency is achieved by incrementally identifying the useful partial or complete derivations and inconsistent combinations of values for variables and recording them as truth maintenance tables to use in future computations (as described in Chapter 4); identifying and recording unextendable partial explanations to avoid rediscovery of the same backtrack points; and allowing the user to incrementally distinguish between incompatible solutions (see Chapter 7). A comparison with other techniques suggests that our approach is better suited for CSPs, as it is simple, intuitive, uniform, empirically efficient and expressively adequate. Finally, we conclude that the constraint logic programming (e.g., [179, 180, 181]) may be considered in terms of hypothetical reasoning.

The rest of this chapter is organized in several sections. Section 5.2 describes how CSPs can be represented and evaluated in the Theorist hypothetical reasoning framework. Section 5.3 establishes a correspondence between the notion of network consistency [101] and consistency trees in our framework. Section 5.4 compares our approach with ATMS-based constraint solving. Section 5.5 discusses the relationship with constraint logic program-

[2]This is an extension to the Theorist hypothetical reasoning framework [136, 134] which is essentially a combination of deduction and abduction. However, Bibel [7] also viewed constraint satisfaction as a problem of logical deduction.

ming systems. Section 5.6 concludes this chapter and gives some suggestions for future research.

5.2 Constraints in Hypothetical Reasoning

5.2.1 Representing the CSP

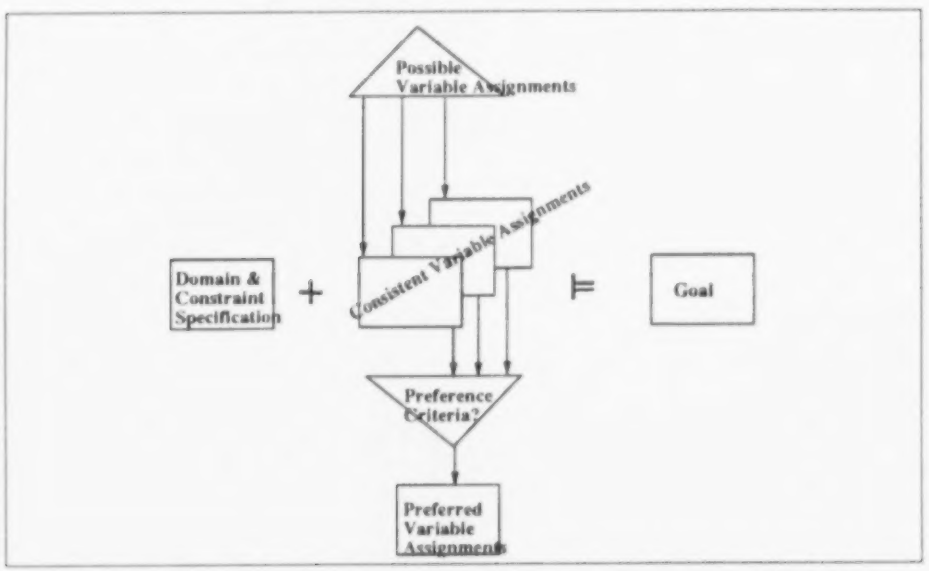


Figure 14: Constraints solving in hypothetical reasoning

To represent CSPs under the framework of hypothetical reasoning, we need to specify constraint relations over variables, including restrictions on their domains. The variables should range over their respective finite domains, rather than the Herbrand universe. Our knowledge may be viewed as knowledge about constraints (i.e., what is not allowed or what is allowed) and domains of variables, which we represent as **facts**. Since a variable can be tentatively assigned any value from its domain, such a value can be viewed as a hypothesis. So, assignments of values to variables are considered as instances of **hypotheses** under our framework of hypothetical reasoning. Each

CSP solution requires a hypothesis with respect to each variable such that instances of that hypothesis range over the domain of the variable.

Figure 14 represents a constraint solving framework using hypotheses. The constraint satisfaction problem is to construct a proof tree for **Goal** using knowledge about constraints and consistent variables assignments. A consistent set of values instantiating the variables in **Goal** forms a solution for the CSP. If **Goal** has all ground instances, the constraint solver verifies whether the given **Goal** is a solution or not. While dealing with multiple possible variable assignments for a given **Goal**, some preference criteria may be used to select between competing variable assignments.

Let us consider a simple case of the n-queens puzzle [101, 67], where n is 4 (see Example 5.1). A queen can take any position on a 4×4 checker board, where rows and columns are labeled from 1 to 4. The constraints disallow two queens on the same row, column, or diagonal. The predicate $queen(X, Y)$ means that a queen has the position (X, Y) on the board; the predicates $xdomain(X)$ and $ydomain(Y)$ are true if X and Y are members of their respective finite domain. The predicate $member(X, D)$ means that $X \in D$.

The constraint satisfaction problem is to find a consistent assignments of positions to the four queens. To solve this problem, the hypothetical reasoning system is asked to explain the goal:

explain $queen(X1, Y1), queen(X2, Y2), queen(X3, Y3), queen(X4, Y4)$.

The response would be:

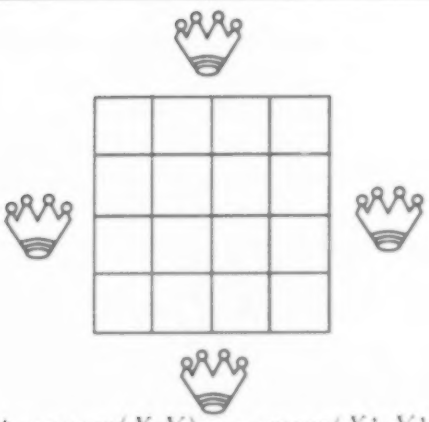
[Answer]: $[queen(1, 2), queen(2, 4), queen(3, 1), queen(4, 3)]$.

[Theory]: [hyp(1,2),hyp(2,4),hyp(3,1),hyp(4,3)].

5.2.2 Incremental Consistency Check

To illustrate the role of consistency tree (see Def. 3) as a useful technique for solving constraints, let us consider a simple problem of constraint satisfaction [147] as shown in Example 5.2.

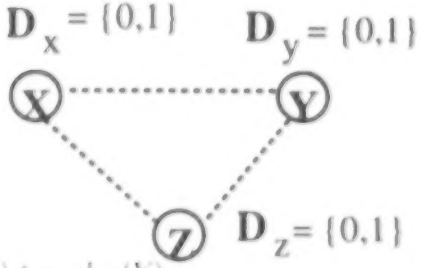
Given X, Y and Z are variables and the respective domains are $D_X = \{0, 1\}$, $D_Y = \{0, 1\}$ and $D_Z = \{0, 1\}$. There are three functions $e_1(X)$, $e_2(Y)$, and $e_3(Z)$. These functions represent some arbitrary expensive computations; additional constraints are $e_1(X) \neq e_2(Y)$ unless $X = Y$, and



fact $\neg queen(X,Y) \leftarrow queen(X1,Y1) \wedge X = X1 \wedge Y \neq Y1.$
fact $\neg queen(X,Y) \leftarrow queen(X1,Y1) \wedge X \neq X1 \wedge Y = Y1.$
fact $\neg queen(X,Y) \leftarrow queen(X1,Y1) \wedge |X - X1| = |Y - Y1|.$
fact $xdomain(X) \leftarrow member(X,[1,2,3,4]).$
fact $ydomain(Y) \leftarrow member(Y,[1,2,3,4]).$
hypothesis hyp(X,Y): $queen(X,Y) \leftarrow xdomain(X) \wedge ydomain(Y).$

Example 5.1: The Four-Queens Problem

$e_2(Y) \neq e_3(Z)$ unless $Y = Z$. We represent this problem in hypothetical reasoning framework as facts and hypotheses (see Example 5.2).



fact $e_1(X) = e_2(X)$.
fact $e_2(X) = e_3(X)$.
fact $var_domain(0)$.
fact $var_domain(1)$.

fact $\neg(e_1(X) = e_2(Y)) \leftarrow xvalue(X) \wedge yvalue(Y)$.
fact $\neg(e_2(Y) = e_3(Z)) \leftarrow yvalue(Y) \wedge zvalue(Z)$.
hypothesis $hyp_xvalue(X) : xvalue(X) \leftarrow var_domain(X)$.
hypothesis $hyp_yvalue(X) : yvalue(X) \leftarrow var_domain(X)$.
hypothesis $hyp_zvalue(X) : zvalue(X) \leftarrow var_domain(X)$.

Example 5.2: A simple CSP

To explain an observation $xvalue(X) \wedge yvalue(Y) \wedge zvalue(Z)$, the constraint solver has to prove all subgoals by instantiating the variables using values from their domains. The first subgoal cannot be proved by facts, however, it can be proved by using hypothesis instance $hyp_xvalue(1)$. Now consistency of $hyp_xvalue(1)$ has to be checked. Figure 15(a) shows that negation of $hyp_xvalue(1)$ can not be proved, so it is consistent. Figure 15(b) shows how a finite failure branch can be extended to form a consistency tree, as defined above. Figure 15(b) shows that $Facts \wedge \{hyp_yvalue(1)\} \vdash \neg hyp_xvalue(1)$, that is, $hyp_xvalue(1) \wedge hyp_yvalue(1) \vdash false$; and $\{hyp_xvalue(1), hyp_yvalue(1)\}$ is an inconsistent variable assignment.

5.2.3 Forward Checking + Truth Maintenance

The motivation for using consistency tree in solving constraints is to extract information about inconsistent sets of hypothesis instances, which can be used to reduce the search space in an *a priori* fashion; and to record the partial explanations and complete proofs to avoid redundant computations. The incremental consistency test enables the constraint solver to identify

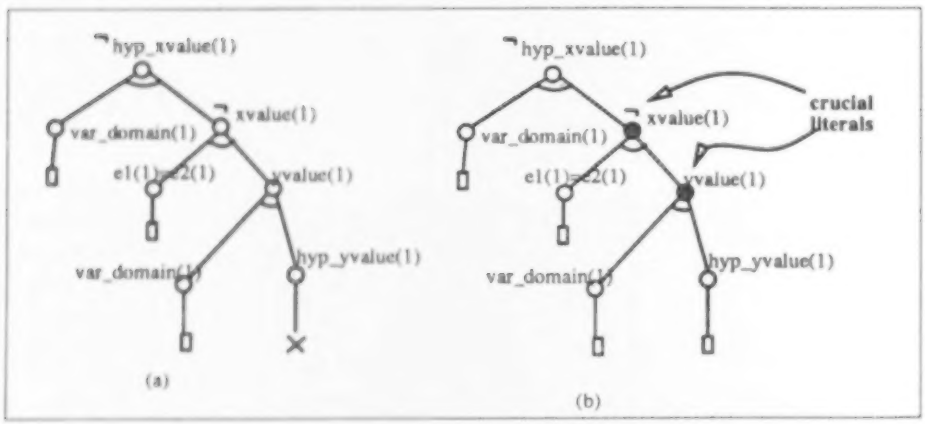


Figure 15: Extending failure tree to consistency tree

inconsistent variable assignments exactly in the same way as *forward checking* does.

Forward checking is one of the search strategies based on active use of constraints for pruning the unnecessary search space in an *a priori* way by removing all variable assignments that are inconsistent with the current variable assignment. Haralick and Elliot [67] have shown experimentally and analytically that forward checking is the better heuristic to solve the CSPs than most of the other consistency based approaches to CSPs with respect to the cost of preprocessing and backtracking. The following theorem establishes a relationship between the efficient search achieved by the *forward checking* [67] and by the consistency tree:

Theorem 24 *Let $X_1, ..., X_n$ be variables; $D_1, ..., D_n$ be finite domains associated with these variables; F is a set of constraints including specification of the finite domains; and $H_{X_1}, ..., H_{X_n}$ be hypotheses with respect to variables. Suppose, a variable X_i is instantiated with a value $x_i \in D_i$, then each inconsistent set of elements w.r.t. x_i, say $\{y_1, ..., y_k\}$ s.t. each $y_j \in D_j$, where $i, j, k \leq n$, is eliminated by the forward checking iff*

$$F \cup \{H_{y_1}, ..., H_{y_k}\} \vdash \neg H_{x_i}.$$

Proof: To prove this theorem, we would simply require to show that consistency tree for a hypothesis instance computes inconsistent combinatio

of past and present choices (or already instantiated variables) with future choices (uninstantiated variables). ■

A truth maintenance system, when embedded with any reasoning system, dynamically records and updates the incremental information supplied by the inference engine. The recorded information is then used by the same inference engine for efficient search while attempting to solve the future reasoning tasks. Here, the truth maintenance component records the useful information extracted from the consistency trees (see Chapter 4), e.g., *potential crucial literals* (Def. 8) and *nogoods* (Def. 14).

In Figure 15(b), nodes indicated by the dark circles are potential crucial literals and recorded as useful information in Table 5.1. The union of the entries in the columns *inset* and *outset* gives a nogood. If *inset* is empty, then the corresponding potential crucial literal is a logical consequence of the facts (Theorem 16), so need not be re-derived.

Potential Crucial Literal	*inset*	*outset*
$\neg xvalue(1)$	$\{hyp_yvalue(1)\}$	$\{hyp_xvalue(1)\}$
$yvalue(1)$	$\{hyp_yvalue(1)\}$	$\{hyp_xvalue(1)\}$

Table 5.1: T_{PCL}: a table of potential crucial literals

5.2.4 Unextendable Partial Explanations

Similar to *forward checking*, for a variable assignment (i.e., an hypothesis instance) *consistency tree* can identify the values in the domains of other variables (i.e. possible hypothesis instances) which are inconsistent with that variable assignment. These inconsistent combinations of variable assignments are computed in terms of *nogoods* which may used to eliminate inconsistent values from domains of the variables in the similar way as forward checking does in CSPs [67]. However, this strategy cannot avoid the backtracking completely.

While constructing an explanation for a given goal if we realize at a certain point that the partial explanation can not be extended (due to wrong choices of values for variables), we record such a partial explanation as an

incompatible set (IS) of hypotheses instances (i.e., variable assignments) with respect to the current goal. This information will avoid re-discovery of the same backtrack points, which will further improve the search efficiency. These *incompatible sets* may be viewed as reasons for dead-ends [31].

Definition 25 (Incompatible set) *A set of hypotheses instances is said to be an incompatible set if it forms a partial explanation for a given goal such that it can not be further extended.*

The *incompatible* sets (ISs) are recorded in a tabular form. The table contains tuples $< g, T_p >$, where g is a goal and T_p is set of hypothesis instances—unextendable partial explanation. For each g the tuple $< g, T_p >$ is minimal if T_p is minimal w.r.t. set inclusion. Minimality of *incompatible sets* is very crucial as regards to the search efficiency.

Theorem 26 *Let $< g, T_p >$ be a tuple containing a goal formula and unextendable partial explanation. Suppose F is a set of facts. Then $F \cup \{g \wedge T_p\} \vdash false$.*

Proof: Since T_p can not be extended further to construct an explanation for the goal g, $F \cup T_p \nvdash g$. Therefore, $F \cup \{g \wedge T_p\} \vdash false$. ■

If a partial explanation T_p for an observation g is an incompatible set, then $\{T_p \wedge g\}$ is inconsistent. In other words, there is no way to prove the goal g with the set of choices made in the set T_p. Therefore, for the given constraint satisfaction problem such incompatible sets may be used as *nogoods*.

Let us consider the four queens problem again (see Example 5.1). Suppose the goal is:

explain queen(X1,1)& queen(X2,2)& queen(X3,3)& queen(X4,4).

If the constraint solvers proves the two subgoals using the hypothesis instances $hyp(1,1)$ and $hyp(3,2)$, then there is no consistent position left for the third queen or the third goal cannot be proved, so it has to backtrack. Here the incompatible set is $\{hyp(1,1), hyp(3,2)\}$. However, this is not the minimal one, later the constraint solvers finds that $\{hyp(1,1)\}$ was the main reason for backtrack points. All of its supersets must be avoided while searching for solutions to this goal. The *minimality* of the incompatible set is very important to improve the search efficiency. The *minimal*

incompatible sets are analogous to the new nogoods derived from the initial nogoods in ATMS-based constraint solving system [27].

We don't treat *incompatible sets* as global nogoods. Rather, these sets used as *nogoods* with respect to the particular CSP. Because, under the same system of constraints, the incompatible set may not be inconsistent for some other CSP. For example, asking consistent positions only for two queens on 4×4 checker board.

5.2.5 Selecting Better Solutions

While solving constraint satisfaction problems we are concerned with: searching for a single solution; or searching for all possible solutions. However, in the worst case there might be exponentially large number of possible variable assignments (i.e., multiple solutions, possible incompatible). In the real world problem, the user is often interested in the preferred set of solutions (e.g., best diagnoses). Here, we propose integration of the idea of *crucial literals* (see chapter 7) with the hypothetical reasoning-based constraint solver to incrementally distinguish between incompatible solutions, that helps the constraint solver to keep the total number of solutions manageable.

A crucial literal is a formula which forms the basis for performing an experiment which seeks to determine the crucial propositions' truth value. That result discriminates the two explanations and provides new knowledge about the world.

In Table 5.1, $yvalue(1)$ is a *crucial literal* in the tuple

$$< yvalue(1), \{hyp_yvalue(1)\}, \{hyp_xvalue(1)\} >.$$

The truth value of this crucial literal not only selects one of the two incompatible solutions, but also asserts a new *nogood*. If $yvalue(1)$ is true, then it will assert $\{hyp_xvalue(1)\}$ as a nogood, which would eliminate all future solutions containing this nogood.

5.2.6 An Incremental Search Procedure

In this section, we specify an incremental search procedure that unifies various heuristics for efficiently searching for single or multiple solutions for the given CSP. When constructing an explanation, if a subgoal is unified with

some literal of the table such that the respective *inset* $\neq \{\}$, then we must check consistency of the elements of *inset* and mark *outset* as a local nogood.

The *local nogood* (see Def. 18) is a conditional and only true for the particular explanation being constructed. However, the same nogood may be a part of another solution for the same problem. The following is a slightly modified version of the search procedure for incrementally constructing explanation as described in chapter 4.

Definition 27 (incremental constraint solver) *Suppose O is an observation sentence, F is a set of facts, and H is a set of hypotheses. Constructing explanations for O requires determination of all success branches of the AND/OR derivation tree $F \cup H \cup \{O\}$. Given a table of potential crucial literals $< pcl, T_{in}, T_{out} >$, a table of minimal incompatible sets $< g, T_p >$ and a set of minimal nogoods NGS, an incremental search procedure for constructing each success branch of this tree is defined as follows:*

1. *Initialize $NGS_{local} := \{\}$, and $T_{Partial} := \{\}$.*
2. *The root node O, is a goal, which is not the successor of any node.*
3. *Node A is a leaf node if A unifies with some element of some potential crucial literal, say pcl, of a tuple $< pcl, \{\}, T_{out} >$.*
4. *Node A is a leaf node if A unifies with some potential crucial literal, say cl, of a tuple $< cl, T_{in}, T_{out} >$ such that $T_{in} \neq \{\}$, and each element of T_{in} is consistent[3]. Then, $T_{Partial} := T_{Partial} \cup T_{in}$ and $NGS_{local} := NGS_{local} \cup \{T_{out}\}$.*
5. *If a node is a conjunction of subgoals, say $\leftarrow A_1, ..., A_i, ..., A_n$, then find a rule $A \leftarrow B_1, ..., B_m$ such that A_i and A are unifiable, then the successor node is $(A_1, ..., A_{i-1}, B_1, ...,$ $B_m, A_{i+1}, ..., A_n)\theta$, where θ is a most general unifier of A_i and A.*
6. *If A is a goal and $\neg A$ is an ancestor of A, then A has no descendant (by reductio ad absurdum).*

[3]Here, consistency means for all $t_i \in T_{in}$, $F \cup T_{Partial} \cup \{T_{in} - t_i\} \nvdash \neg t_i$, and there is no $ng \in NGS \cup NGS_{local}$ such that $ng \subseteq T_{Partial}$.

Also, we have to check that for each $< g, T_p >$, where $g \subseteq O$, such that $T_p \not\subseteq T_{partial}$.

7. *If A is a goal which unifies with an instance of hypothesis* $h_i \in H$ *and* $F \cup T_{Partial} \not\vdash \neg h_i$, *there is no* $ng \in NGS \cup NGS_{local}$ *such that* $ng \subseteq T_{Partial} \cup \{h_i\}$, *then A has no descendant.* $T_{Partial} := T_{Partial} \cup \{h_i\}$.
8. *Nodes which are empty goals have no descendants.*
9. *If A is a goal, and no step from 3 to 9 is true (i.e., A can not be explained —* $T_{partial}$ *is unextendable), then* $ISs := ISs \cup \{< O, T_{Partial} >\}$.
10. *When a proof tree is found,* $Explanation := T_{Partial}$.

If we want to search for all possible consistent variable assignments or solutions for a given CSP, the crucial literals may be used to incrementally discriminate between incompatible solutions. Figure 16 gives a a pictorial description of the incremental constraint solving.

Let us again consider the four-queens (see Example 5.1) problem to illustrate the function of incremental constraint solving procedure. Suppose the goal is

$$\textbf{explain}\ queen(X1, Y1) \wedge queen(X2, Y2) \wedge queen(X3, Y3) \wedge queen(X1, Y1).$$

To construct an explanation the constraint solver has to instantiate variables using consistent assignments (i.e., hypothesis instances). If the search is performed in depth-first chronological order, the first goal to be proved is $queen(X1, Y1)$. The variables $X1$ and $Y1$ are instantiated with the first value from domain using hypothesis $hyp(1, 1)$. Obviously, the first task is to test consistency of this hypothesis instance by attempting to prove its negation. Figure 17 shows that there nine are possible consistency trees for $hyp(1, 1)$.

The nodes indicated by dark circles are potential crucial literals. After constructing each consistency tree, the incremental search procedure extracts useful information from that tree. For example, Table 5.2 is the table of potential crucial literals indicates that the set $\{hyp(1, 1), hyp(1, 2)\}$ is *nogood* and the literals $xdomain(1), ydomain(1)$ and $ydomain(2)$ are consequences of facts alone, and, thus, need not to be re-derived (i.e., the membership test is not repeated). The literals $\neg queen(1, 1)$ and $queen(1, 2)$ may be used as crucial between two possible solutions for this CSP.

As shown in Figure 17, the table derived from consistency tree (1) is used in computation of consistency tree (2) and so on. The size of the table grows

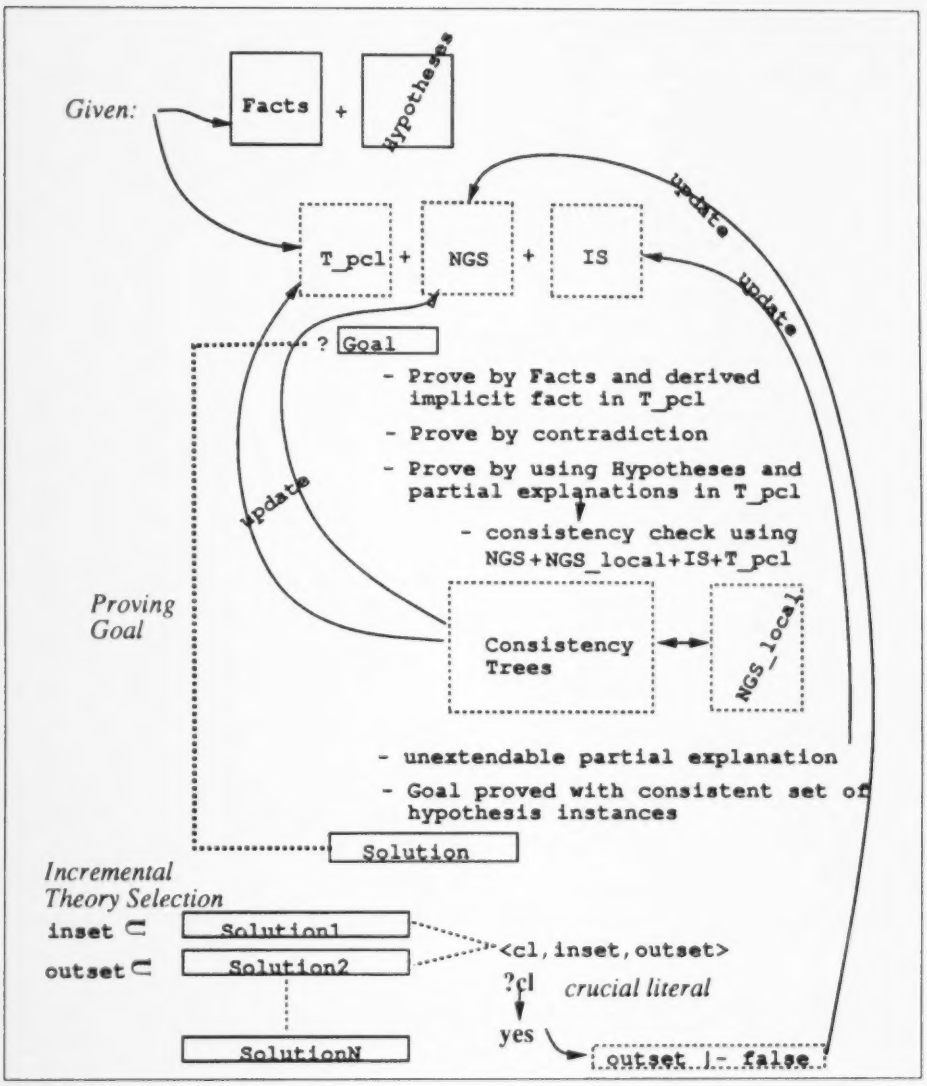


Figure 16: Incremental Constraint Solving

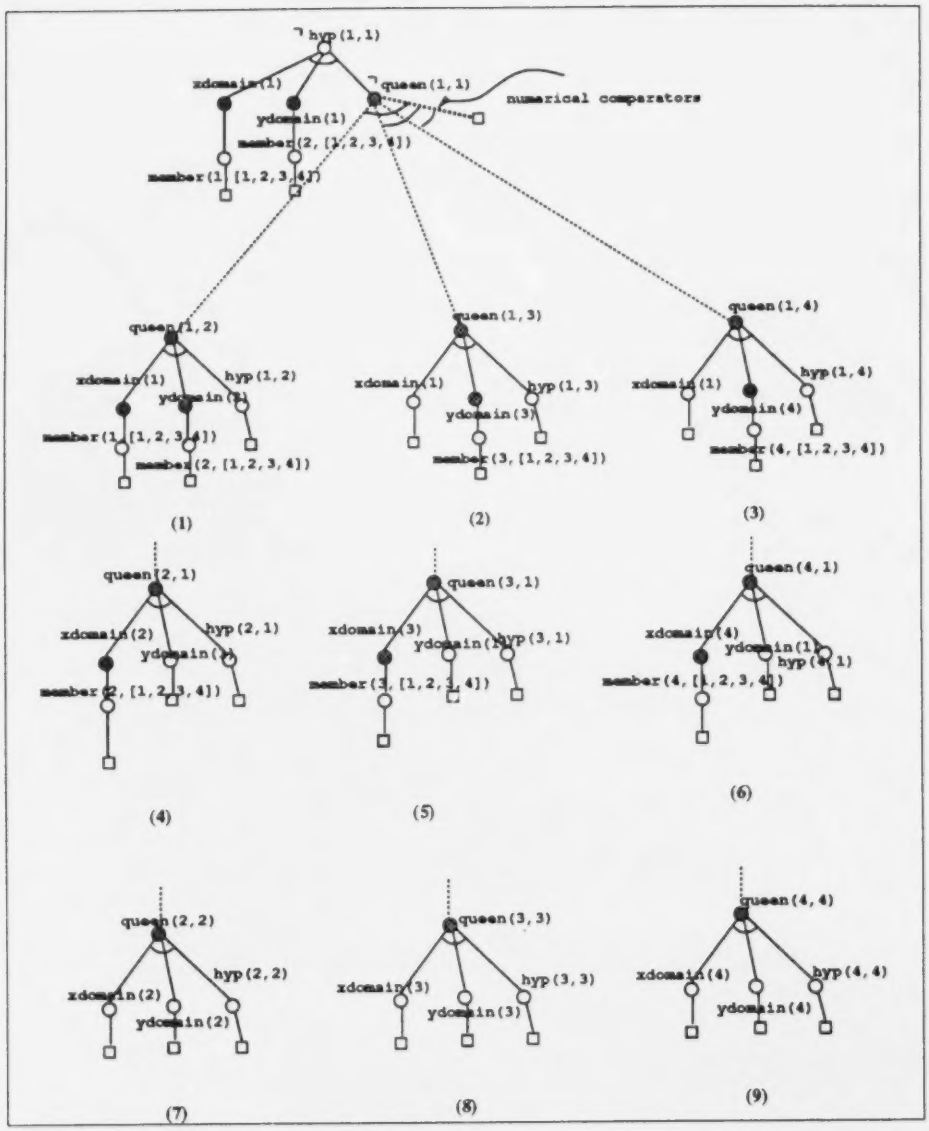


Figure 17: Consistency trees in Four-Queens problem

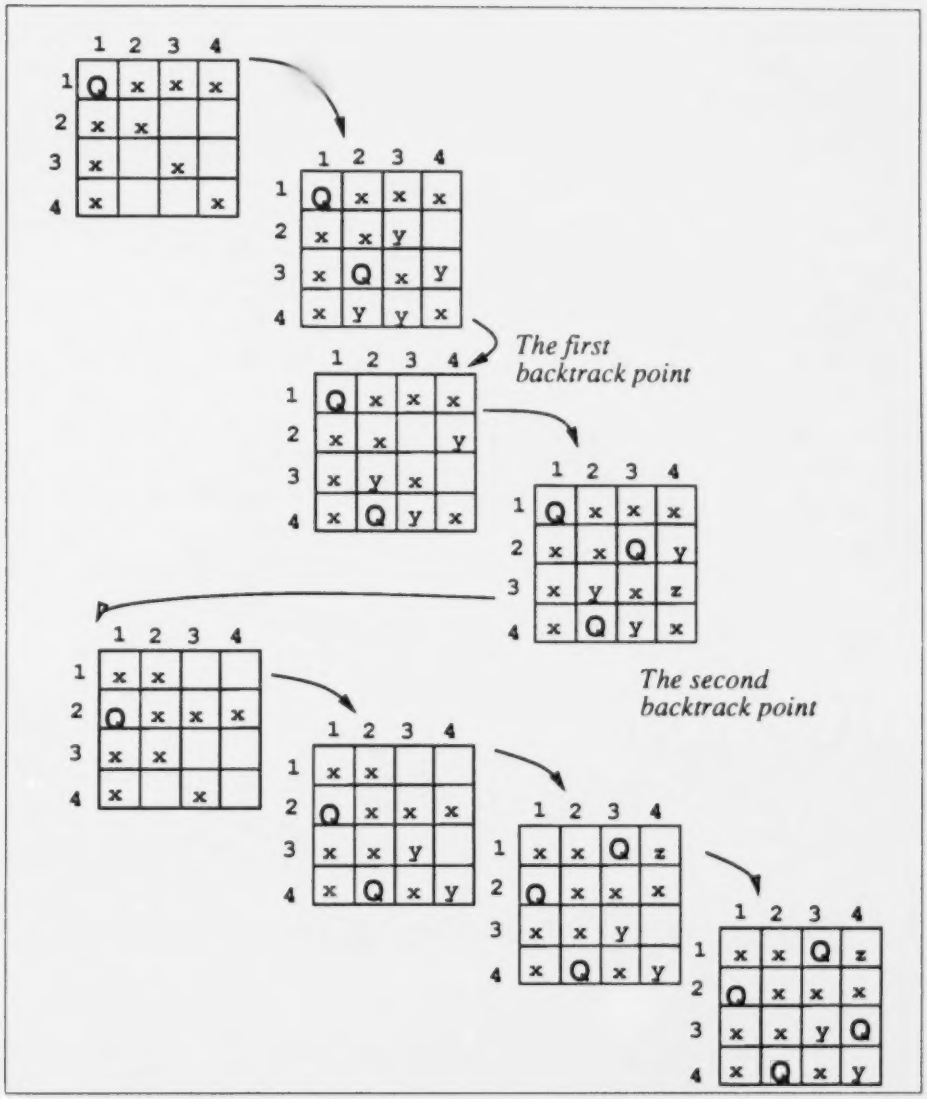


Figure 18: State Transitions in Four-Queens problem

Potential Crucial Literal	*inset*	*outset*
$xdomain(1)$	$\{\}$	$\{hyp(1,1), hyp(1,2)\}$
$ydomain(1)$	$\{\}$	$\{hyp(1,1), hyp(1,2)\}$
$ydomain(2)$	$\{\}$	$\{hyp(1,1), hyp(1,2)\}$
$\neg queen(1,1)$	$\{hyp(1,2)\}$	$\{hyp(1,1)\}$
$queen(1,2)$	$\{hyp(1,2)\}$	$\{hyp(1,1)\}$

Table 5.2: A table of potential crucial literals in Four-Queens problem

monotonically which in turn reduces the future search space. In the consistency trees (7),(8), and (9) the membership test is not at all required. The total number of consistency trees for finding the first solution are 26. Figure 18 shows how a consistent assignment of variables is achieved. The boxes filled with x, y and z indicate positions that become inconsistent after first (i.e., $hyp(1,1)$), second, and third choice respectively. Here, the constraint solvers backtracks 2 times. The resulting *unextendable partial explanations* w.r.t. the $goal = \{queen(X1,Y1), queen(X2,Y2), queen(X3,Y3), queen(X4,Y4)\}$ are:

$< goal, \{hyp(1,1), hyp(2,3)\} >$
$< goal, \{hyp(1,1), hyp(2,4), hyp(3,2)\} >$

Obviously, the *minimal incompatible set* is $\{hyp(1,1)\}$. We record this as the reason for the two dead-ends. The constraint solver will avoid all supersets of this minimal incompatible set as solution for the given CSP. That helps to reduce the search space for future solutions. In this case, the second solution can be found without any backtracking and with 6 consistency trees; and two more consistency check will determine that there is no more solution exists.

5.3 Network Consistency

To show a relationship between consistency techniques for CSPs and the hypothetical reasoning framework, we must note that many researchers in the constraint satisfaction paradigm view a CSP as a network of relations.

Each node represents a variable and a finite domain of elements which can be assigned to the variable. In the framework of hypothetical reasoning, a node represents a hypothesis associated with a domain over which instances of that hypothesis can range. When a system of constraints with n variables is represented as a network of n nodes, a consistent instance of the network forms a solution to the CSP. We will show that finding a consistent instance of the network is equivalent to the task of constructing an explanation in hypothetical reasoning.

In the following, we define Mackworth's *node-consistency*, *arc-consistency*, and *path-consistency*, and Freuder's *k-consistency* from the perspective of hypothetical reasoning. Our definitions capture intuitions similar to those mentioned in [45]; and are similar to the definitions adopted by De Kleer while comparing ATMS with CSP techniques in [27]. We use the Example 5.3, taken from [45], to illustrate the notion of network consistency under hypothetical reasoning.

Let us consider variables X_1, X_2 and X_3. The respective domains are $D_1 = \{a, b\}$, $D_2 = \{e, f\}$ and $D_3 = \{c, d, g\}$. Binary constraints are $C_{12} = \{be, bf\}, C_{13} = \{bc, bd, bg\}$ and $C_{23} = \{ed, fg\}$. The binary constraints restrict combinations of values from two different domains. Example 5.3 shows representation of this problem in hypothetical reasoning.

5.3.1 Node-consistency

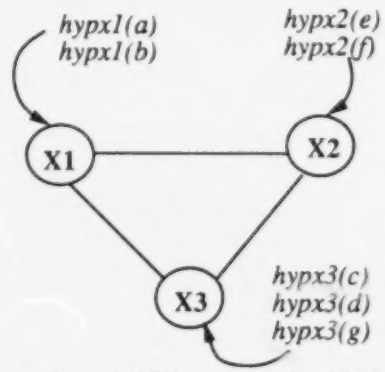
Node consistency ensures that the instances of each hypothesis $h_i(X)$ for node i range over the domain of the variable X. In other words, node consistency eliminates all illegal values from domains of the variables. The hypothetical reasoning system checks node consistency by attempting to prove negation of the respective hypothesis instances.

Definition 28 *Suppose each node i of a network is associated with a hypothesis $h_i(X)$, where the variable X has a finite domain. Then the network is node-consistent iff*

$$\forall x \in X \;\; F \not\vdash \neg h_i(x).^4$$

[4] F is a set of clauses representing constraint relations and domain of the each variable.

fact $\neg x1value(a) \leftarrow x2value(e).$
fact $\neg x1value(a) \leftarrow x2value(f).$
fact $\neg x1value(a) \leftarrow x3value(c).$
fact $\neg x1value(a) \leftarrow x3value(d).$
fact $\neg x1value(a) \leftarrow x3value(g).$
fact $\neg x2value(e) \leftarrow x3value(c).$
fact $\neg x2value(e) \leftarrow x3value(g).$
fact $\neg x2value(f) \leftarrow x3value(c).$
fact $\neg x2value(f) \leftarrow x3value(d).$



fact $x1domain(X) \leftarrow member(X, [a, b]).$
fact $x2domain(X) \leftarrow member(X, [e, f]).$
fact $x3domain(X) \leftarrow member(X, [c, d, g]).$
hypothesis $hypx1(X) : x1value(X) \leftarrow x1domain(X).$
hypothesis $hypx2(X) : x2value(X) \leftarrow x2domain(X).$
hypothesis $hypx3(X) : x3value(X) \leftarrow x3domain(X).$

Example 5.3: A constraint network

In Example 5.3, each node represents a variable and corresponding possible hypotheses instances that can be used to assign a value to that variable. The domains of variables may be viewed as ground instances of hypotheses associated with the nodes. If for a node i, $\forall h_i(x) \in h_i(X)$, s.t. $F \not\vdash \neg h_i(x)$ is true, then all element x can be assigned to the variable. In the above example, we have hypothesis instances $hypx1(a)$ and $hypx1(b)$ for node $X1$; $hypx2(e)$ and hypx2(f) for node $X2$; and $hypx3(c), hypx3(d)$ and $hypx3(g)$ for node $X3$.

5.3.2 Arc-consistency

An arc is viewed as a pair of nodes, say $h_i(X)$ and $h_j(Y)$. If an instance of one node does not make an instance of other node inconsistent, then the pair of hypotheses instances are said to be mutually consistent. Arc consistency ensures that every pair of nodes has atleast one consistent instance. The inconsistent pairs of hypotheses instances may be recorded as nogoods.

Definition 29 *Suppose each pair of nodes i and j is represented by a pair of hypotheses $(h_i(X), h_j(Y))$, where X and Y range over finite domains. Then the network is arc-consistent iff*

$$\forall x \exists y, \text{ s.t. } F \cup h_i(x) \not\vdash \neg h_i(y), \text{ where } x \in X \text{ and } y \in Y.$$

In hypothetical reasoning framework, arc consistency for a pair of nodes i and j eliminates a combnation of hypothesis instances, say $\{h_i(x), h_j(y)\}$ if

$$\forall x \exists y, \text{ s.t. } F \cup h_i(x) \vdash \neg h_j(y), \text{ where } x \in X \text{ and } y \in Y.$$

For example, the hypothesis instances $hypx3(c)$ and $hypx1(a)$ are eliminated when we force the constraint network to be arc-consistent. Logically, $Facts \cup \{hypx3(c)\} \vdash \neg hypx1(a)$ (i.e., $\{hypx1(a), hypx3(c)\}$ is a nogood).

In other words, if the following is true,

$$\exists x \exists y, \text{ s.t. } F \cup h_i(x) \vdash \neg h_j(y), \text{ where, } x \in X, y \in Y.$$

then consistency checking identifies the set $\{h_i(x), h_j(y)\}$ as nogood. In Example 5.3, pairs of instances $\{hypx2(e), hypx3(g)\}$ and $\{hypx2(f), hypx3(d)\}$ form nogoods.

5.3.3 Path-consistency

Let us consider three nodes i, j and k. The corresponding hypotheses are: $h_i(X), h_j(Y)$, and $h_k(Z)$ such that X, Y and Z range over finite domains. If the path formed by nodes i, j and k is consistent, then the pairs of nodes i and j, and nodes j and k are arc-consistent.

Definition 30 *Suppose $h_i(X), h_j(Y)$, and $h_k(Z)$ are hypotheses associated with nodes i, j and k, where X, Y and Z range over finite domains. Then the network is path-consistent iff*

if $\forall x \exists z$ s.t. $F \cup h_i(X) \not\vdash \neg h_k(z)$; and $\forall z \exists x$ s.t. $F \cup h_k(Z) \not\vdash \neg h_i(x)$, then
$\forall x \forall z \exists y$ s.t. $F \cup h_i(X) \not\vdash \neg h_j(y)$ and $F \cup h_k(Z) \not\vdash \neg h_j(y)$. Where $x \in X, y \in Y$ and $z \in Z$.

In the example shown in Example 5.3, the path-consistency for nodes $X1$, $X2$ and $X3$ results in only two possible variable assignments. The hypothetical reasoning-based constraint solver can be asked to solve the constraint satisfaction problem of assigning consistent values to the variables $X1$, $X2$ and $X3$, as follows:

explain $x1value(X1) \wedge x2value(X2) \wedge x3value(X3)$.

The response would be:

[Answer]: $\{x1value(b), x2value(e), x3value(d)\}$
[Theory]: {hypx1(b),hypx2(e),hypx3(d)}
[Answer]: $\{x1value(b), x2value(f), x3value(g)\}$
[Theory]: {hypx1(b),hypx2(f),hypx3(g)}
No more Answers.

5.3.4 k-consistency

The above definitions can be generalized in terms of *k-consistency*. Suppose each node of the constraint network is represented by a hypothesis. Then a network of size n consists a set of hypotheses, say $H = \{h_1(X_1), ..., h_n(X_n)\}$.

Definition 31 *Let $H_k = \{h_1(X_1), ..., h_k(X_k)\}$ be any subset of H, where $k \leq n$. Then the network is k-consistent iff*

$\forall_{j \neq i} x_j \exists x_i$ *s.t.* $F \cup \bigcup_{j=1}^{k} h_j(X_j) - \{h_i(X_i)\} \not\vdash \neg h_i(x_i)$, *where* $x_i \in X_i$, $x_j \in X_j$ *and* $i, j \leq k$.

Obviously, node consistency, arc consistency and path consistency are special cases of k-consistency (i.e., $k = 1$, $k = 2$ and $k = 3$ respectively).

5.3.5 What's a solution for a CSP?

The objective of the different levels of consistency mentioned above is to find one consistent instance, or all consistent instances of the constraint network. A consistent instance is a solution. The framework of hypothetical reasoning can compute a consistent instance of the network relation of size $k \leq n$ as an explanation when that relation is viewed as an observation sentence.[5] The following theorem establishes a relationship between generating explanations in the hypothetical reasoning framework and finding consistent instances of network:

Theorem 32 *Let $X_1, ..., X_n$ be variables, and their respective finite domains be $D_1, ..., D_n$. Let F be a set of clausal sentences representing knowledge about the world including domains of variables and constraint relations; H be a set of hypotheses, say $\{h_1(X_1), ..., h_n(X_n)\}$, where each $h_i(X_i)$ corresponds to the node representing the variable X_i. Suppose a constraint relation over the subset of the variable is $c_k = (X_1, ..., X_k)$, where $k \leq n$, and each variable in this relation can be instantiated using some hypothesis instance, (i.e., $c_k(X_1, ..., X_k) \leftarrow \bigwedge_{i=1}^{i=k} h_i(X_i)$ is a fact in the database). Then there is a consistent assignment of variables involved in constraint c_k iff*

$F \cup E \models c_k(X_1, ..., X_k)$.
Where, E ($\subseteq H$) is a consistent set of hypotheses instances—an explanation.

[5] The search procedure in our system can be made to follow any of the consistency-based CSP techniques.

In the above theorem, if the goal sentence is ground, then our system can verify whether the given solution satisfies the system of constraints. If there are variables in the goal sentence, then the system generates each possible explanation as a solution to the given CSP.

5.4 ATMS-based Approach to CSPs

The ATMS [26] is an assumption-based truth maintenance system primarily aimed at overcoming the limitations of the classical truth maintenance system TMS [37] by manipulating assumptions rather than justifications. The input to the ATMS consists a set of propositional literals and clauses derived from them, and a distinguished set of assumptions. The main tasks of the ATMS are to find all minimal nogoods (i.e., inconsistent sets of assumptions) and to identify for every literal a set of environments (sets of assumptions) in which it holds. The ATMS has several advantages over TMS: requires no backtracking; considers multiple contexts simultaneously which may be mutually contradictory; simplifies complex machinery of the TMS; does not need global consistency; etc. A comparison of ATMS with CSP techniques is given in [27].

In the ATMS framework, instances of each variable are represented by propositional literals; an explicit assumption is required that a variable can not have more than one value; and constraints are specified in terms of what is not allowed. Let us again consider the Example 5.3 (which is also considered in [27]). The following are the propositional clauses representing variables and their possible values:

$x1 : a \vee x1 : b$
$x2 : e \vee x2 : f$
$x3 : c \vee x3 : d \vee x3 : g$
$\neg x1 : a \vee \neg x1 : b$
$\neg x2 : e \vee \neg x2 : f$
$\neg x3 : c \vee \neg x3 : d$
$\neg x3 : c \vee \neg x3 : g$
$\neg x3 : d \vee \neg x3 : g$

The binary constraints are:

$\neg x1 : a \vee \neg x2 : e$
$\neg x1 : a \vee \neg x2 : f$
$\neg x1 : a \vee \neg x3 : c$
$\neg x1 : a \vee \neg x3 : d$
$\neg x1 : a \vee \neg x3 : g$
$\neg x2 : e \vee \neg x3 : c$
$\neg x2 : e \vee \neg x3 : g$
$\neg x2 : f \vee \neg x3 : c$
$\neg x2 : f \vee \neg x3 : d$

The ATMS problem is to identify all nogoods and list them as initial nogoods: { $\{x1:a, x1:b\}$, $\{x2:e, x2:f\}$, $\{x3:c, x3:d\}$, $\{x3:c, x3:g\}$, $\{x3:d, x3:g\}$, $\{x1:a, x2:e\}$, $\{x1:a, x2:f\}$, $\{x1:a, x3:c\}$, $\{x1:a, x3:d\}$, $\{x1:a, x3:g\}$, $\{x2:e, x3:c\}$, $\{x2:f, x3:c\}$, $\{x2:f, x3:d\}$, $\{x2:e, x3:g\}$ }. Then it resolves upon them to derive new nogoods. Here, for instance $\{x3:c\}$ is a new nogood, and add it to the nogoods set. Since nogoods are maintained as minimal sets w.r.t. set inclusion, smaller new nogood may reduce the total number. In our approach, we do not have to compute all nogoods, rather nogoods are incrementally computed and recorded as needed. For this example, if the problem is formulated as

explain $x1value(X) \wedge x2value(Y) \wedge x3value(Z)$.

The hypothetical reasoning system will find the first solution by constructing 6 consistency trees (i.e., only computing six nogoods). The ATMS notion of new nogood seems to be a problem specific, rather than a global nogood. We record such nogoods as reasons for dead-ends encountered while searching for a solution to the specific problem. We refer them as *incompatible sets*. The advantage of our style is that if the goal statement slightly changes for the same system of constraints, then the *incompatible set* may no longer be a nogood. For example, $x1value(a)$ is incompatible set for the goal requiring consistent values for $x1value(X1), x2value(X2), x3value(X3)$. But, it is not nogood under the same system of constraints for the goal $x1value(X1)$. We have seen similar cases in the queens problems as well.

However, the following observations indicate the main limitations of the ATMS-based constraint solver:

- Since the ATMS has to assume explicitly that each variable can have at most one value from its domain, exponential number of clauses is required to express this assumption.

- It can only compute all solutions when the observation sentence is a single atom [147], which is too weak to represent a realistic constraint relation.

- The ATMS has to compute and list all of the minimal nogoods which is quite expensive. That is, more consistency checks are required to avoid backtracking which seems to be similar to the *full lookahead* search

strategy. Obviously, this methodology is not good for the problems requiring a single solution.

- Although, theorem proving in the ATMS framework is decidable, Selman in his recent paper [164] suggests apart from the fact that the ATMS may have to compute all solutions (which may be exponential in general), finding one of them may require exponential time. Therefore, even after these restrictions on the language, the ATMS can not guarantee computational tractability.

- The basic ATMS does not support nonmonotonicity. It may be good to compute all solutions in some restricted problem domain. But, it does not provide any mechanism to incrementally discriminate among the competing solutions. The only way is to find out all solutions and then apply some theory preference criteria. Since, in general, the number of solutions are exponentially large, the selection process may be very expensive particularly in the diagnostic reasoning.

In comparison to the ATMS, the hypothetical reasoning-based approach seems to overcome most of the above limitations. Our system is committed to provide an adequate expressive power using full clausal logic. The unification mechanism in the underlying hypothetical theorem proving system instantiate a variable by at most one value from its domain at a time. Consequently, there is no need to explicitly specify the exponential number of clauses. The consistency tree in our system is equivalent to *forward checking* [156], which incrementally computes nogoods and record them. Forward checking is a nice compromise between consistency checks and backtracking.

5.5 Constraint Logic Programming

The subject of integrating constraint satisfaction paradigm within the logic programming languages has been a hot topic in logic programming and AI around the world for the last few years (e.g., CHIP [179, 180, 181], constraint logic programming scheme (CLP) [81], Prolog III [18], Trilogy [182], CAL [2]). The main motivation for developing all of these Horn clause logic-based constraint solving systems is to actively use constraints to reduce the unnecessary search space in an *a priori* way, in contrast to the standard

backtracking which prunes the space in an *a posteriori* way (i.e., after having discovered a failure). Here is a summary of the existing constraint logic programming systems:

- Prolog III [18] was originated from Prolog II [17], it is designed to solve a system of equality constraints on boolean terms and a system of linear equations and inequalities on rational terms. Its search procedure uses a saturation method combined with SL-resolution.

- CHIP a constraint handling in Prolog [179, 180, 181], extends logic programming by using ideas of consistency-based constraint satisfaction techniques [101, 45, 67] to actively use constraints; and by introducing a notion of domains to restrict instances of the variables involved in a CSP to be finite. Accordingly, the unification mechanism is modified.

- CLP [81] is an extension of the logic programming language scheme. Instead of unification, it uses a concept of constraint solving. The efficiency of CLP relies on its constraint-solver which can identify a satisfiable set of constraints; and uses incremental mechanism to avoid re-recomputing the same solution.

- CAL [2] is a constraint logic programming language developed at ICOT. CAL modifies the criteria of selecting the computation domain in CHIP, such that solving constraints should be decidable. Then it finds a canonical form of a constraint which is an answer.

- Trilogy [182] is a constraint logic programming language which extends the computation domain to integers, basically used to solve numerical constraints on natural numbers.

- Echidna [69] is yet another constraint logic programming system.

In comparison to these constraint logic programming systems, our hypothetical reasoning system is expressively more powerful, and provides an opportunity to integrate truth maintenance and other heuristics for constraint solving. For example, in the constraint logic programming system CHIP [179, 180, 181] for finding a single solution of the Four-Queens problem requires 2 backtracks and 34 consistency checks. However, we have illustrated in section 5.2.6 that our approach requires 2 backtracks and 26 consistency

trees. We believe that the Prolog implementation of our formalism may be viewed as a constraint logic programming system which does not require modifications or extensions to the syntax or semantics of the basic logic program. Rather, the deduction process of Prolog is controlled by the meta-knowledge specified at the meta-level, in the spirits of [70, 55, 9, 8].

5.6 Conclusion

This chapter has shown how constraint satisfaction problems can be specified and efficiently solved under the general framework of hypothetical reasoning. We have established a relationship between consistency based CSP techniques and consistency check in the hypothetical reasoning system. The computational efficiency of our approach to hypothetical reasoning relies on incrementally compiling inconsistencies, recording proofs as potential crucial literals, recording unextendable partial explanations as incompatible sets, and using crucial literals to discriminate between two incompatible solutions (if required).

Constraint logic programming languages (e.g. [181]) and assumption-based truth maintenance system (ATMS) [26] are closely related to our work. Since both ATMS and constraint logic programming system use propositional Horn clause logic, our approach is more powerful in specifying a system of constraints. The Prolog implementation of our system may be viewed as a constraint logic programming system which does not require changes in syntax and semantics or unification mechanism of the basic logic program.

Chapter 6

Meta-reasoning: An Incremental Compilation Approach

6.1 Introduction

A meta-level reasoning system is a system that performs reasoning about reasoning. In the conceptualization of meta-knowledge, expressions, subexpressions and sequences of expressions are treated as objects of the domain of discourse, and functions and relations are defined over these objects [56, Chapter 10]. The primary motivation for using meta-level reasoning as the basis of intelligent systems, is to have a declarative language with high expressive power which provides the capability of explicitly and separately representing control knowledge. The idea is to describe control knowledge that can be the basis for reasoning about strategies at the meta-level, in order to make the deductive process more computationally efficient [70, 55, 8]. For example, in a blocks world, specifying the physical movement of a robot arm would be at object level, and the specification of all planning task (i.e., how to decide an action to be performed at object level) would be at meta-level [55].

While meta-programming has been widely accepted as a tool to explicitly specify control, the overhead involved in interpretive execution of an object program by the meta-interpreter makes a meta-reasoning system computa-

tionally inefficient. Recent research on meta-programming techniques suggest that the overhead problem can be avoided by using compilation or partial evaluation (e.g., [6, 30, 97, 175]). In his recent paper [97], Lloyd discussed a number of research problems related to meta-logic programming. Two of these problems are: how to efficiently implement a meta-logic programming system via a compilation process, and how to dynamically update a compiled knowledge base.

This chapter considers meta-reasoning from the perspective of THEORIST [136, 134]. Section 6.2 describes how Extended-THEORIST can be used as a powerful tool for meta-logic programming. In section 6.3, an incremental compilation[1] approach is proposed to avoid some of the efficiency problems of meta-reasoning. Section 6.4 proposes a method to update the dynamically changing knowledge base. Our update method does not require re-compilation of knowledge base or retraction of facts; and it minimizes the number of assertions stored using the table of potential crucial literals.[2] Section 6.5 establishes a relationship between Reiter and de Kleer's [147] *prime implicants* and potential crucial literals, which shows that incremental compilation may be quite useful in computing and managing prime implicants. Finally, we conclude the chapter.

6.2 THEORIST: A Meta-reasoner

In our framework of hypothetical reasoning, the hypotheses *names* provide a convenient method to record instantiation of arbitrary clauses. We show that this naming facility provides a basis to specify control information for efficient deduction.

Let us consider an example [70] of the theory of integers consisting of the constant 0, the successor function s and the predecessor function p. Using these functions, the specification of an algorithm to compute the sum of two

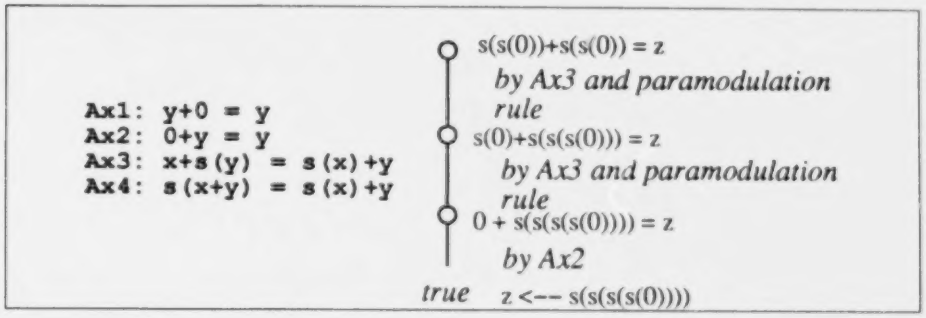
[1] As the domains of discourse may be infinite, it is impossible to determine the complete set of possible hypotheses instances, so inconsistencies can not be predetermined. Therefore, the only way to improve computational efficiency of such a rich system is to incrementally compute and record useful information which can be used to guide future search.

[2] Surprisingly, this idea turns out to be very useful for developing a rational belief revision framework [152].

integers is as given below:

$add(n, m) = m$, if $n = 0$
$add(n, m) = add(p(n), s(m))$, otherwise.

The following are the axioms and the proof tree used for computing the sum of two integers:



Example 6.1: Computing sum of two integers

The above proof tree appears to very small, but if the axioms Ax3 and Ax2 are used the wrong way round, the theorem prover would generate an infinite search space. Hayes [70] shows that this non-deterministic behavior of the reasoner may be transformed into a deterministic one by explicitly specifying some control primitives. For example, in this case a naming convention may be used: $A : u$ means that A must appear at the top line and u must be instantiated in the end; $t1 = t2 :\leftarrow$ means that $t1$ replaces $t2$; $t1 = t2 :\rightarrow$ means $t2$ replaces $t1$. Thus, the following specifications of control information makes the algorithm deterministic.

$Ax2 :\rightarrow$, $Ax3 :\leftarrow$ and $s(s(0)) + s(s(0)) = z : z$

We can recast the above problem in the Theorist representation as follows:

In the above specification, $Ax2$ and $Ax3$ are treated as hypotheses. The hypothesis $lax2$ is equivalent to $Ax2 :\rightarrow$ and $rax2$ is equivalent to $Ax2 :\leftarrow$. The control information ($Ax2 :\rightarrow$, $Ax3 :\leftarrow$ and $s(s(0)) + s(s(0)) = z : z$) can easily be specified under the above framework by asserting {rax2} and {lax3} as nogoods, i.e., restricting the use of hypotheses $lax3$ and $rax2$.

To solve this problem, we ask Theorist to explain

fact solve(plus(X,Y),Z) ← replace(plus(X,Y),Z) & sfun(Z).
fact solve(plus(X,Y),Z) ← replace(plus(X,Y),Z1) & solve(Z1,Z).
hypothesis lax2: replace(plus(0,Y),Y).
hypothesis rax2: replace(Y,plus(0,Y)).
hypothesis lax3: replace(plus(X,s(Y)),plus(s(X),Y)).
hypothesis rax3: replace(plus(s(X),Y),plus(X,s(Y))).

Figure 19: Theorist Database

$$solve(plus(s(s(0)), s(s(0))), Z)$$

which is the goal statement (i.e., appears at the top line of the proof tree). The predicate $sfun(Z)$ (treated as built-in predicate) is true if Z is an integer, i.e., the variable Z can be instantiated. In addition, the Theorist system gives a theory to explain the result—a set of hypotheses (or axioms) used.

In Meta-Theorist [62], an extension of Theorist is proposed to introduce higher level commands for specifying theory pruning and preference criteria.

6.3 Incremental Compilation

The advent of logic programming systems such as Prolog was primarily motivated to provide an efficient logical system with high expressive power, and a simple elegant semantics. The expressive power enables users to write their programs in a natural way quite easily and quickly. A simple and elegant semantics makes the tasks of debugging, testing and verification very simple. However, most of the current Prolog systems consider a subset of first-order logic, i.e., Horn clauses, and have only a depth first, left to right inference strategy. Therefore, despite of deductive capability, Prolog like systems are quite inefficient as regards to their computational performance and flexibility of control.

While meta-level reasoning has been shown to be a very useful technique for solving the problems of Prolog systems, a new problem arises due to a substantial overhead in handling object-level variables while meta-level interpreter executes the object program. This problem could be avoided if we compile the meta-level specifications into the object program, and then

executes the compiled code directly and efficiently on current Prolog systems.

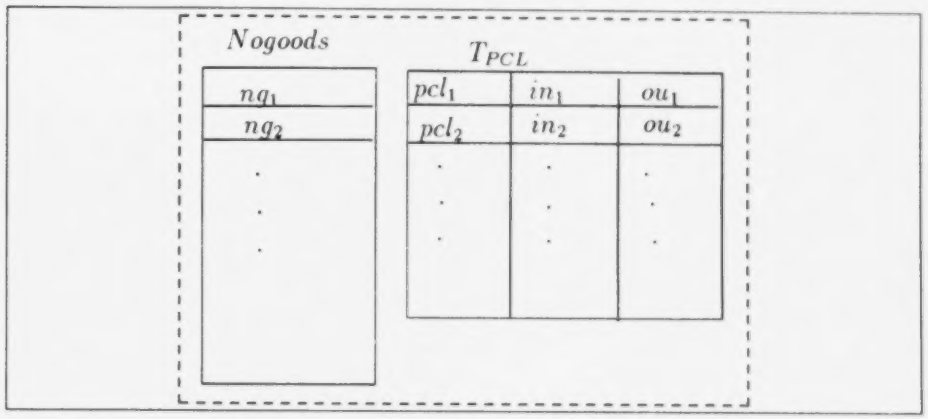


Figure 20: The extra knowledge accumulated incrementally

However, both interpretive and compilation approaches are complementary with respect to their advantages and disadvantages [175]. The interpretive approach is less efficient, but allows user to understand the behavior of the inference mechanism which can be used to modify the control. On the other hand, compilation approach is efficient, but it is difficult to understand the compiler and the compiled code. The partial evaluation technique suggested in [175], is an attempt to take advantages of both of these approaches without sacrificing the efficiency. Here, we present yet another alternative *incremental compilation* to deal with the problem of computational efficiency.

The incremental compilation process proceeds as follows. First, the compiler translates meta-level specification of facts and hypotheses into two sets of sentences of clausal logic: Π_{facts} and $\Pi_{hypotheses}$. Next, it computes inconsistent instances of hypotheses — nogood — and potential crucial literals during each consistency check. Figure 20 illustrates this extra information that is accumulated incrementally while constructing explanations. NGS is a table of minimal nogoods and T_{PCL} is a table of tuples $< pcl_i, in_i, ou_i >$ where pcl_i is a potential crucial literal, in_i and ou_i are inset and outset w.r.t. pcl_i. NGS and T_{PCL} incrementally updated as described in Chapter 4.

6.3.1 Translating Meta-level Specification

The following steps would be required to translate the internal representation into sets of sentences of clausal logic: [3]

- Each fact $fact(H, B)$ is treated as an implication $H \leftarrow \bigwedge_{i=1}^{i=n} b_i$, where each $b_i \in B$. All contrapositives of $H \leftarrow \bigwedge_{i=1}^{i=n} b_i$ are asserted as clauses. These clauses form the set Π_{facts}.
- Each hypothesis $hypothesis(Name)$ is translated into a clause $Name \leftarrow$ with empty body. These clauses form the set $\Pi_{hypothesis}$.

Example 6.2 illustrates this simple translation process.

$$\left[\begin{array}{l} \mathbf{F} = \left\{ \text{ fact } l_1 \leftarrow l_2 \wedge ... \wedge l_n. \right\} \\ \mathbf{H} = \left\{ \text{ hypothesis } h1(X). \right\} \end{array}\right]$$

$$\Downarrow$$

$$\left[\begin{array}{l} \mathbf{\Pi_{facts}} = \left\{ \begin{array}{l} l_1 \leftarrow l_2 \wedge ... \wedge l_n. \\ \neg l_2 \leftarrow \neg l_1 \wedge ... \wedge l_n. \\ ... \\ \neg l_n \leftarrow \neg l_1 \wedge l_2 \wedge ... \wedge l_{n-1}. \end{array} \right\} \\ \mathbf{\Pi_{hypothesis}} = \left\{ \; h1(X). \; \right\} \end{array}\right]$$

Example 6.2: Translating meta-sentences into object-level code

6.3.2 Computing Inconsistencies and Potential Crucial Literals

To compute inconsistent combinations of hypothesis instances and potential crucial literals, we use the *consistency tree* (see Def. 3 in Chapter 4). The idea is that during the test for consistency of a hypothesis instance we can identify some combinations of hypothesis instances that lead to contradictions, i.e., *nogoods*. We can also record partial derivations in the form of a table of potential crucial literals.

[3] The preprocessor must have a translator for compiling input $wffs$ into CNF form.

$$
\left[\begin{array}{l}
\mathbf{F} = \left\{\begin{array}{l}
\textbf{fact } g \leftarrow p1(a) \wedge p2(a) \\
\textbf{fact } p1(a) \leftarrow f(a) \wedge h1(a) \\
\textbf{fact } \neg p1(a) \leftarrow p2(a) \wedge h2(a) \\
\textbf{fact } p2(a) \leftarrow f(a) \wedge h3(a) \\
\textbf{fact } f(a) \leftarrow r(b) \\
\textbf{fact } r(b)
\end{array}\right\} \\
\mathbf{H} = \left\{\begin{array}{l}
\textbf{hypothesis } h1(X) \\
\textbf{hypothesis } h2(X) \\
\textbf{hypothesis } h3(X)
\end{array}\right\}
\end{array}\right]
$$

$$
\Downarrow
$$

$$
\left[\begin{array}{l}
\mathbf{\Pi_{facts}} = \left\{\begin{array}{l}
g \leftarrow p1(a) \wedge p2(a). \\
\neg p1(a) \leftarrow \neg g \wedge p2(a). \\
\neg p2(a) \leftarrow \neg g \wedge p1(a). \\
p1(a) \leftarrow f(a) \wedge h1(a). \\
\neg f(a) \leftarrow \neg p1(a) \wedge h1(a). \\
\neg h1(a) \leftarrow \neg p1(a) \wedge f(a). \\
\neg p1(a) \leftarrow p2(a) \wedge h2(a). \\
\neg p2(a) \leftarrow p1(a) \wedge h2(a). \\
\neg h2(a) \leftarrow p1(a) \wedge p2(a). \\
p2(a) \leftarrow f(a) \wedge h3(a). \\
\neg f(a) \leftarrow \neg p2(a) \wedge h3(a). \\
\neg h3(a) \leftarrow \neg p2(a) \wedge f(a). \\
f(a) \leftarrow r(b). \\
\neg r(b) \leftarrow \neg f(a). \\
r(b).
\end{array}\right\} \\
\mathbf{\Pi_{hypotheses}} = \left\{\begin{array}{l}
h1(X). \\
h2(X). \\
h3(X).
\end{array}\right\}
\end{array}\right]
$$

Example 6.3: Compiling Theorist Database

Let us consider an example (Example 6.3) to illustrate the consistency tree and the information that can be derived from it.

To explain an observation g, the theorem prover has to prove the two subgoals $p1(a)$ and $p2(a)$. If $p1(a)$ is selected first, then it would require an hypothesis instance $h1(a)$. To use $h1(a)$ in a proof tree, the unprovability of $\neg h1(a)$ (see Figure 21(a)) has to be established. The consistency tree is then formed by extending the failure subtree as shown in Figure 21(b).

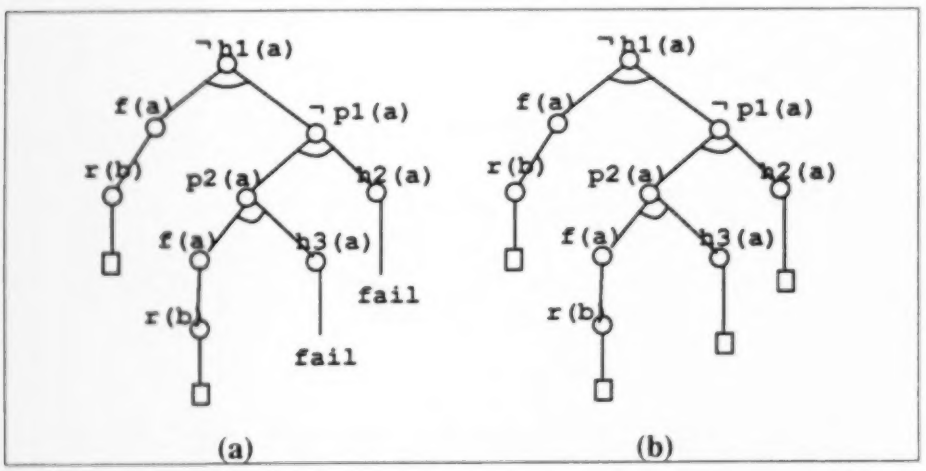


Figure 21: Extending failure tree to consistency tree

Figure 21(b) shows that $Facts \models h3(a) \wedge h2(a) \rightarrow \neg h1(a)$ (i.e., $Facts \cup h1(a) \wedge h2(a) \wedge h3(a) \vdash false$). Thus $\{h1(a), h2(a), h3(a)\}$ is called a *nogood*. Nogoods (NGS) are maintained as sets minimal w.r.t. set inclusion.

Other information extracted from the Figure 21(b) is recorded in a tabular form, which is denoted by T_{PCL}, as shown in Table 6.3. Each tuple in the table consists of $< pcl, inset, outset >$.

In Chapter 4, we have described algorithms to incrementally compute nogoods and potential crucial literals. We then specify the update methods for NGS and T_{PCL}. Here, we reemphasize that the consistency tree for an hypothesis instance h_i is an extension to the finitely failed subtree of the AND/OR tree $\Pi_{facts} \cup \{\neg h_i\}$, using new instances of hypotheses from $\Pi_{hypotheses}$. Since there may be an infinite number of hypotheses, we can not predetermine all hypothesis instances and their corresponding consistency

Potential Crucial Literal	*inset*	*outset*
$f(a)$	$\{\}$	$\{h1(a), h2(a), h3(a)\}$
$p2(a)$	$\{h3(a)\}$	$\{h2(a), h1(a)\}$
$\neg p1(a)$	$\{h3(a), h2(a)\}$	$\{h1(a)\}$

Table 6.3: $Table_{PCL}$

trees. Our pragmatic approach relies on constructing consistency trees for only those hypotheses instances which are required to form explanations of an observation.

6.4 Updating the Compiled Knowledge

The compiled knowledge consists of the object-level sentences (translated by the static component) and extra information that is incrementally accumulated in each consistency search. Now our problem is to update the compiled knowledge base when a new fact is learned. One simplistic solution is to add the new knowledge at meta-level and recompile. That results in losing all the previously determined extra information and redoing the translation of meta-level specification into object-level code. In many AI applications (e.g., diagnostic reasoning), where the knowledge about the domain of discourse changes frequently, the recompilation approach makes the advantages of compilation ineffective and adversely affect the overall efficiency of a reasoning system.

Our knowledge base in the hypothetical reasoning framework consists of facts and hypotheses. We assume that the new knowledge does not contradict the facts, only consistency status of some hypotheses or combinations of hypotheses instances may be changed or some hypotheses instances might become obsolete. Under this assumption, the update process has to perform two major tasks: first, it computes the effects of a new fact in terms of inconsistencies (i.e., nogoods), and then add them to the current set NGS; second, to update the table of potential crucial literals. The new knowledge

is compiled and its object-code may be asserted into the set Π_{facts} if it does not appear in T_{PCL}[4].

Algorithm 33 *Suppose we learn a new fact, say f. Then the following steps are used to update the knowledge base* [5]

1. *If there is some $< pcl, inset, outset > \in T_{PCL}$ such that $f \equiv pcl$ and $inset$ is $\{\}$, then it is redundant to add f to the Π_{facts}.*
2. *If there is some $< pcl, inset, outset > \in T_{PCL}$ such that $f \equiv pcl$ and $inset \neq \{\}$, then*
   (a) *$inset := \{\}$.*
   (b) *add outset to NGS such that minimality w.r.t. set inclusion is preserved.*
   (c) *All tuples $< pcl', inset', outset' >$, where $inset \subseteq inset'$, $\Pi'_{facts} \wedge pcl \cup inset' - inset \models pcl'$, and $\Pi_{facts} \equiv \Pi'_{facts} \wedge pcl$, are replaced by $< pcl', inset' - inset, outset' >$.* [6]
3. *If none of the above steps are true, then construct a consistency tree for $\neg f$ using Π_{facts} and $\Pi_{hypotheses}$, and the Def. 3. Add all T_i to NGS (using Algorithm 21) such that $\Pi_{facts} \cup T_i \vdash \neg f$, where $T_i \subseteq \Pi_{hypotheses}$. Add f to the compiled knowledge.*

Theorem 34 *Let Π_{facts} and $\Pi_{hypotheses}$ be sets of facts and hypotheses at the object level, $< pcl, inset, outset >$ is a tuple in T_{PCL} and f is a new fact. Then the following holds:*

1. *if $f \equiv pcl_k$ and $inset = \{\}$, then it is redundant to add f to Π_{facts};*
2. *if $f \equiv pcl_k$, then outset is a nogood; and*

---

[4] Here, we should note that no *retraction* is required and *assertions* are minimized by using the table of Potential Crucial Literals.

[5] Here, we assume that $F \not\vdash \neg f$. See Makinson [104] for a survey of the logical issues in logics of theory change.

[6] If we assume that each hypothesis instance supports a unique atomic sentence, then this process becomes simple — all insets, S, which are supersets of *inset* will be replaced by $S - inset$. I.e., the set of hypotheses instances *inset* has become obsolete in the view of new fact.

3. *if* $\Pi_{facts} \cup T_i \vdash \neg f$, *where* $T_i \subseteq \Pi_{hypotheses}$, *then* T_i *is a nogood.*

Proof:

1. Since $\Pi_{facts} \cup inset \models pcl_k$ and $inset = \{\}$ and $pcl_k \equiv f$, f has already been inferred as a theorem from the facts. So, it is redundant to add f to Π_{facts}.
2. Since $pcl_k \equiv f$, $\Pi_{facts} \cup f \models \neg outset$. Therefore, *outset* is inconsistent—nogood.
3. By definition of nogood and 2. ■

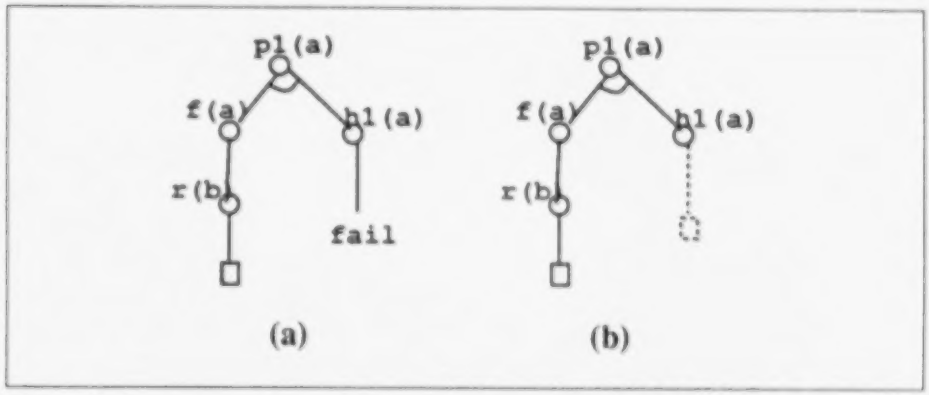
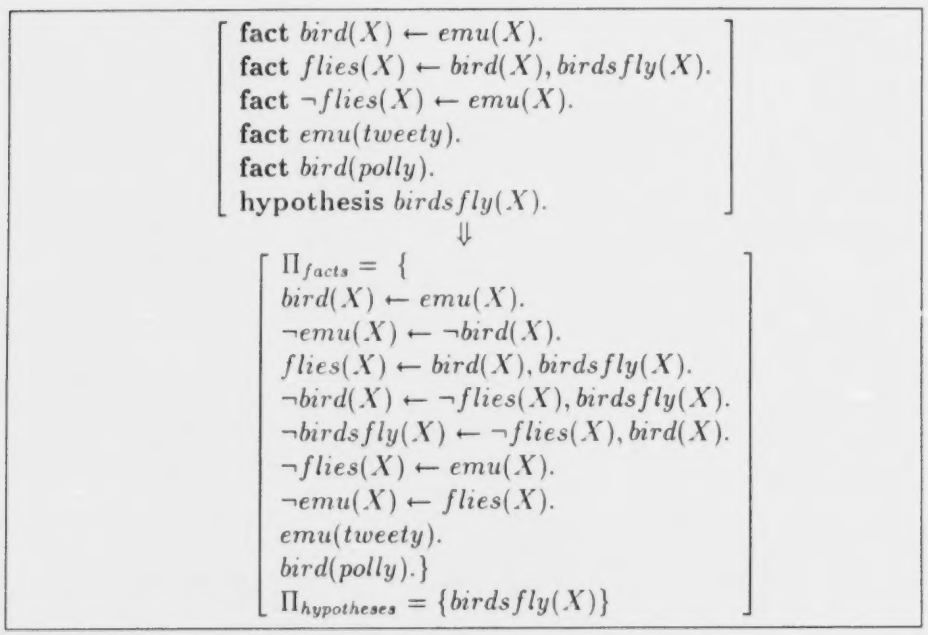


Figure 22: Identifying nogoods caused by $\neg p1(a)$

Suppose our current knowledge is the same as described in Example 6.3. If we learn a new **fact** $\neg p1(a)$, we use Algorithm 33 to compute its effects in terms of nogoods. Its negation $p1(a)$ can not be derived from Π_{facts} (Figure 22(a)), but can be derived if we use an instance of hypothesis $h1(a)$ (Figure 22(b)). Thus, $\{h1(a)\}$ is new nogood. However, we do not have to actually construct a proof for $p1(a)$. Because, the new fact is an element of the table of PCLs (see Table 6.3) which indicates that $\{h1(a)\}$ is nogood if $\neg p1(a)$ is true. When $\{h1(a)\}$ is added to NGS, the set $\{h1(a), h2(a), h3(a)\}$ is replaced by $\{h1(a)\}$ as a minimal nogood. Now the observation g can not be explained on the basis of updated compiled knowledge.

Let us consider another example that represents a classic problem of non-monotonic reasoning. The facts and hypotheses, and their compilation are shown in Example 6.4.



Example 6.4: Birds Fly

In this case, there exists finite failure branch in the consistency tree for $\neg birdsfly(polly)$, but there is a success branch for $\neg birdsfly(tweety)$; thus, $birdsfly(tweety)$ is an inconsistent hypothesis instance and $birdsfly(polly)$ is consistent (see Figure 23). However, the finite failure branch of the consistency tree can not be extended using other instances of hypotheses. In other words, there is no hypothesis instance whose combination with $birdsfly(polly)$ will be inconsistent.

From these trees, we have identified a singleton nogood $\{birdsfly(tweety)\}$. Therefore, we can not explain $flies(tweety)$, but $flies(polly)$ can be explained by using the hypothesis instance $birdsfly(polly)$.

Suppose we learned a fact that polly is an emu (i.e., **fact** $emu(polly)$). We then use the update algorithm to determine the effects of the new knowledge.

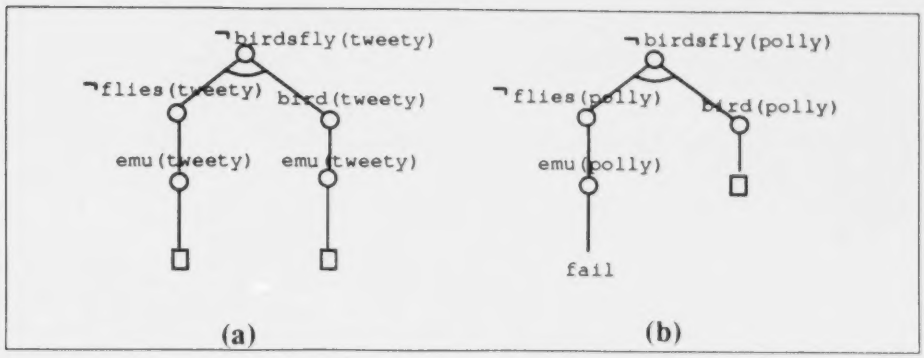


Figure 23: Consistency Trees

Here, $\Pi_{facts} \cup \{birdsfly(polly)\} \vdash \neg emu(polly)$, means the set $\{birdsfly(polly)\}$ is now a nogood. So we can no longer explain either $flies(tweety)$ or $flies(polly)$.

6.5 Relationship with Prime Implicants

Reiter and de Kleer [147] give a generalization of the basic ATMS [26] called the *clause management system* (CMS). The CMS works in cooperation with a *reasoner* which occasionally transmits a clause (it may be a first-order formula) that is recorded by the CMS as a propositional clause. Whenever the reasoner issues a query, the CMS is responsible for computing *minimal supports* or *minimal explanations* for the query.

Definition 35 (Minimal Support [147]) *Let Σ be a set of clauses, and C a clause. A clause S is support clause for C iff:*

$$\Sigma \models S \vee C, \text{ and}$$
$$\Sigma \not\models S.$$

S is a minimal support for C w.r.t. Σ iff there is no support S' for C w.r.t. Σ s.t. $S' \subset S$.

If C is an observation, then $\neg S$ is a simplest conjecture (i.e., minimal explanation w.r.t. set inclusion) which sanctions the conclusion C. Comparing

$\neg S$ with E in Definition 2, if we restrict E to be minimal w.r.t. set inclusion then $\neg S$ is exactly the minimal E [7].

The CMS is concerned with two main tasks: to efficiently compute minimal supports and to dynamically update the knowledge base. Reiter and de Kleer [147] discussed the trade-off between the efficiency of computing explanations and the update with respect to the interpretive and the compiled approaches. Both tasks can not be performed efficiently in any one of the two approaches. For efficient computation of minimal explanations the CMS stresses that the set of clauses in Σ should not be kept as they are transmitted from the reasoner, rather their prime implicants (i.e., compiled knowledge) should be recorded.

Definition 36 (Prime Implicant [147]) *A prime implicant of a set Σ of clauses is a clause C such that*

$$\Sigma \models C, \text{ and}$$
$$\nexists C' \text{ s.t. } C' \subset C \text{ and } \Sigma \models C'.$$

Reiter and de Kleer have shown that there is no loss of information if Σ is represented in terms of its prime implicants. However, the problem of updating the prime implicants in the light of new clauses received from the reasoner remains difficult [147]. The following theorems establish a relationship between the incrementally complied knowledge and the prime implicants.

Theorem 37 *Let $< C, In, Out >$ be a tuple in the table of potential crucial literals. Suppose F is a set of facts and H is a set of hypotheses. Then*

1. *$\neg In$ is a support clause for the literal C iff*

   $$F \cup In \models C \text{ and}$$
   $$F \not\models \neg In, \text{ where } In \text{ is a subset of instances of hypotheses from } H.$$

2. *$\neg In$ is a minimal support for C iff there is no support $\neg In'$ for C such that $\neg In' \subset \neg In$.*

[7] The reasoning framework in Definition 2 is a generalised form of abductive reasoning, since C is not restricted to be a single literal. Also, the set of clauses in Σ are divided into two types: facts and hypotheses, which reduces the number of possible explanations for the given obsevation.

Proof: 1. If $\neg In$ is a support for C, then $F \models \neg In \vee C$ (i.e., $F \cup In \models C$) and $F \not\models \neg In$. Similarly, if $F \cup In \models C$ and $F \not\models \neg In$ implies that In is an explanation for the observation C or $\neg In$ is a support for C.

2. Obvious. ■

Theorem 38 *Let $< C, In, Ou >$ be a tuple in the table of potential crucial literals. Suppose F is a set of facts and each literal in F can be used as a hypothesis (i.e., F becomes identical to Σ [147]). Then $\neg In \vee C$ is a prime implicant of F iff $\neg In$ is a minimal support for C.*

Proof: This theorem is similar to the Theorem 3 in [147]. Its proof is obvious from the definitions of minimal support (Def. 35) and prime implicant (Def. 36). ■

Now we have to show that the table of potential crucial literals is complete with respect to the minimal supports for a given literal C.

Theorem 39 *Let C be an internal node (a potential crucial literal) in a consistency tree. Suppose there is no infinite branch of the AND/OR subtree rooted at C. Then the table of potential crucial literals contains all possible minimal supports for the literal C.*

Proof: The AND/OR derivation tree contains three types of branches: *infinite*, *finite failure* and *success branches*:

1. There is no infinite branch (our assumption).

2. If we do not allow all of the literals in F to be considered as hypotheses, then there exists at least one leaf node which can not be a hypothesis. Consequently, it can not be part of an explanation for C. However, if we allow each literal in F to be hypothesis, then the unextendable leaf nodes can be assumed as hypotheses. Therefore, there will be no failure branch.

3. A leaf node of a success branch can be either of the following three types:

   (a) A literal which is a fact or an instance of a fact. Such literals are theorems derived from the facts, so are not part of an explanation.

(b) A literal which contradicts its ancestor. If that ancestor is also an ancestor of C, then we ignore such sucess branch, because the inset is not consistent (see Def. 3). However, if that ancestor is a descendent of C we treat such a node as a fact in (a).

(c) A literal which can be considered as a hypothesis, each of these literals is part of an explanation for the literal C.

Therefore, we conclude that the consistency trees provide complete information, under the above assumptions, to determine all possible minimal supports for a potential crucial literal. ■

Comparing our approach with Reiter and de Kleer's [147] approach, both the table of potential crucial literals and the prime implicants make computation of explanations more efficient by avoiding the re-derivation of proofs. The difference between these approaches is that we restrict a subset of literals be considered as hypotheses, where as Reiter and de Kleer do not. With our extra information, we are generating fewer prime implicants than the other approaches including [147, 77, 84]. Our approach of incremental compilation of potential crucial literals and nogoods, and updating the compiled knowledge may be quite useful, because it is possible to restrict the number of prime implicants to be manageable.[8]

6.6 Conclusion

We have addressed the issues related to the practical efficiency of a meta-reasoning system used for hypothetical reasoning. We proposed a simple and elegant approach that relies on incrementally compiling inconsistencies, recording proofs as potential crucial literals, and incrementally updating the dynamic knowledge base. We have established a relationship between our approach and the generalised ATMS called *clause management system* (CMS) [147]. The relationship shows that our approach may be used as an alternative for the interpreted and compiled approach for computing and updating the *prime implicants*.

[8]The classification of knowledge base as facts and hypotheses allows fewer prime implicants than other approaches. Further, restrictions can be imposed, for example, prespecifying the literals which can be allowed as potential crucial literals. However, more investigation is required.

Chapter 7

Using Crucial Literals to Select Better Theories

7.1 Introduction

Scientific theory formation has long been a classical problem in the philosophy of science. Despite its wide and frequent use by philosophers and scientists, the term scientific theory or rule has been defined only in the form of a *code or rule of craft*. Popper [142] discarded the prevailing ambiguous definitions and suggested that scientific theories are conjectures or tentative hypotheses which may be rejected or accepted on the basis of observations. In other words, a scientific theory is a set of sentences or laws which form an *explanation* for an observation entailed by that theory along with facts. If the observation can be explained by a theory, the theory may be accepted; otherwise it is rejected.

There may be many explanations for a single set of observations; for example, one set of symptoms might be caused by a number of diseases. The obvious problem is how to decide which explanation is better. The reason that multiple theories arise for a given observation is that we do not have enough knowledge to discriminate these theories (cf. [66]). Therefore, in order to discriminate theories, we need to test the theories against more knowledge—that is, learn more about the world. Popper [142] suggests the simple criteria that a theory T_1 is better than theory T_2 if T_1 explains more observations than T_2, and T_1 has passed a test which T_2 has failed to pass.

Using this criteria, Seki and Takeuchi [163] proved that if the two Horn clause theories T_1 and T_2 are incompatible with respect to some integrity constraint IC, then there exists a *crucial literal* whose truth is compatible with only one of T_1 or T_2. An integrity constraint is a fact or a combination of facts which asserts a condition that is false in the problem domain. A literal L is a *crucial literal* with respect to T_1 and T_2 if either T_1 or T_2 explains L, but not both. For Horn clause theories coupled with integrity constraints, Seki and Takeuchi discuss an algorithm that identifies crucial literals, and show how they can be used to discriminate theories.

For a given set of observations, THEORIST may generate multiple explanations. These explanations may be categorized into three forms: first, those which are mutually incompatible; second, those which are mutually consistent but there exists some literal (e.g., *askables* [166]) which may discriminate among them; third, those which are mutually consistent but there is no askable literal which can be used to discriminate them (i.e., all such explanations are incomparable). This set of explanations is based on our current knowledge about the world (i.e., the problem domain), so more knowledge about the world decreases the number of possible explanations. Here our main interest is to deal with the first kind of multiple explanations. Like well-designed scientific experiments, crucial literals guide our pursuit of domain knowledge that distinguishes these explanations.

The integrity constraints used by Seki and Takeuchi [163] and Shapiro [167] are simply facts in Theorist. Shapiro uses an idea similar to crucial literals for finite Horn clause theories; his algorithmic method relies on *oracles* to identify negations of facts. However, our approach of identifying crucial literals is different from both [163] and [167], as we deal with richer theories (first-order logic), and identify as many as possible, rather than one crucial literal for the same pair of incompatible explanations. We integrate the Seki and Takeuchi method with Theorist without using the notion of integrity constraint. The enhanced system identifies a crucial literal between two theories, then the *crucial literal* is posed as a query to the user (cf. [166]). The result of the experiment is asserted as new knowledge.

Since we repeat this process incrementally, the new knowledge will not only discriminate two theories of current interest, but will also potentially eliminate other incorrect theories which may later arise. This can lead to a significant reduction in the total number of explanations. Our implementation consists of a prototype built on top of THEORIST that illustrates

the use of crucial literals in the problems of electronic circuit diagnosis and medical diagnosis. Also, we implement a more general approach to identify crucial literals using *consistency trees* (see Def. 3). The idea of a consistency tree provides the user with greater opportunity (cf. the Seki and Takeuchi method) to minimize the number of explanations for a given set of observations.

The rest of this chapter is organized as follows. Section 7.2 defines the role of crucial literals as experiments. The integration of the Seki and Takeuchi approach is discussed and its use within Theorist is described. Section 7.3 describes how a consistency tree can be used to identify all possible crucial literals. Section 7.4 illustrates the application of crucial literals for incrementally discriminating between incompatible diagnoses. Finally, we conclude the chapter.

7.2 Crucial Literals as Experiments

A key in the process of growth of scientific knowledge, is to perform experiments that help in acquiring more knowledge as well as in resolving ambiguities in the current knowledge. Our interest is to identify experiments that will refute one of two mutually incompatible sets of hypotheses.

The Seki and Takeuchi [163] solution introduces the idea of a *crucial literal*. A literal L is a *crucial literal* between two mutually incompatible theories, say T_1 and T_2, if either of these explains L, but not both. i.e.,

$$Facts \cup T_1 \vdash L \tag{7.1}$$

$$Facts \cup T_2 \vdash \neg L. \tag{7.2}$$

A *crucial literal* is a formula which forms the basis for performing an experiment which seeks to determine the crucial proposition's truth value. That result discriminates the two theories and provides new knowledge about the world. For example, suppose the truth value of literal L is *true*; the preferred theory is that determined by the logical relationship shown in (7.1).

To use crucial literals first requires their identification, and eventually, their value. Here, we are mainly concerned with the first issue. The second task can be left to the user as in [166], or ultimately a set of effectors and receptors.

Seki and Takeuchi [163] describe an algorithm to compute a *crucial literal* between two mutually incompatible Horn clause theories. Because of the expressive weakness of Horn Clause theories, they define incompatibility of two theories with respect to some *integrity constraint*. An *integrity constraint* (IC) is a fact or a set of facts which asserts a condition that is false in the problem domain. Their results can be summarized as follows.

Definition 40 [163] *The theories T_1 and T_2 are said to be incompatible with respect to IC iff*

1. $Facts \cup T_1 \not\vdash IC$
2. $Facts \cup T_2 \not\vdash IC$
3. $Facts \cup T_1 \cup T_2 \vdash IC$

Theorem 41 [163] *There exists a* crucial literal L *between two theories T_1 and T_2 such that $Facts \cup T_1 \models L$ and $Facts \cup T_2 \models \neg L$ if T_1 and T_2 are mutually incompatible with respect to IC.*

Seki and Takeuchi illustrated their method by using an example of circuit diagnosis as shown in Figure 24. This diagnosis problem may be specified under the Theorist framework as shown in Example 7.1.

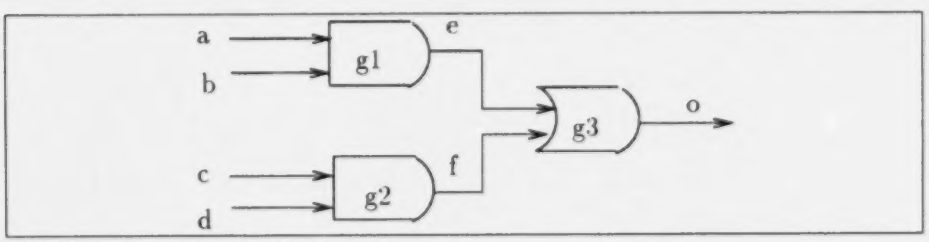


Figure 24: A circuit diagnosis problem

In Example 7.1, the integrity constraint IC is a combination of the contradictory formulas $[node(o, low), node(o, high)]$. An observation $node(o, low)$ can be explained by more than one theories. For example, the theories $[not_ok(g3), not_ok(g2), ok(g1)]$ and
$[ok(g3), not_ok(g2), ok(g1)]$ explain the observation $node(o, low)$. Both of

```
fact and(g1).
fact and(g2).
fact or(g3).
fact node(a,high).
fact node(b,high).
fact node(c,high).
fact node(d,low).
fact not node(X,low) ← node(X,high).
fact not node(X,high) ← node(X,low).
fact and_gate(G,high,high,high) ← and(G), ok(G).
fact and_gate(G,low,high,low)← and(G), ok(G).
fact and_gate(G,high,low,low)← and(G), ok(G).
fact and_gate(G,low,low,low)← and(G), ok(G).
fact and_gate(G,high,high,low)←and(G), not_ok(G).
fact and_gate(G,low,high,high)←and(G), not_ok(G).
fact and_gate(G,high,low,high)←and(G), not_ok(G).
fact and_gate(G,low,low,high)←and(G), not_ok(G).
fact or_gate(G,high,high,high)← or(G), ok(G).
fact or_gate(G,low,high,high)← or(G), ok(G).
fact or_gate(G,high,low,high)← or(G), ok(G).
fact or_gate(G,low,low,low)← or(G), ok(G).
fact or_gate(G ,high,high,low)← or(G), not_ok(G).
fact or_gate(G ,high,low,low)← or(G), not_ok(G).
fact or_gate(G ,low,high,low)← or(G), not_ok(G).
fact or_gate(G ,low,low, high)← or(G), not_ok(G).
fact node(o,O)← node(e,E),node(f,F),or_gate(g3,E,F,O).
fact node(e,E)← node(a,A),node(b,B),and_gate(g1,A,B,E).
fact node(f,F)← node(c,C),node(d,D),and_gate(g2,C,D,F).
hypothesis ok(G).
hypothesis not_ok(G).
```

Example 7.1: THEORIST Database for Circuit Diagnosis

these theories are incompatible with respect to IC. However, the theories $[not_ok(g3), not_ok(g2), ok(g1)]$ and $[not_ok(g3), ok(g2), ok(g1)]$, which also explain the above observation, are incompatible in general, but they are not incompatible with respect to the integrity constraint IC. So this approach based on explicitly specifying integrity constraints identifies only a subset of possible incompatible theories.

The enhanced Theorist framework considers two hypotheses at a time, and analyzes each node of the proof tree for $Facts \cup Theories \cup \{observation\}$ to determine the crucial literals.

Furthermore, the second extension incrementally computes a *crucial literal* for a pair of incompatible theories. The crucial literal is posed as a query to the user (i.e., to perform experiment) and the user's response (i.e., result of the experiment) refutes one of two theories, identifying the other as a preferred theory. The system then asserts the result of the experiment as new fact, which may eliminate many further potential theories which may otherwise be generated.

If an observation is explained by a number of explanations $T_1, T_2, ..., T_n$, then we use the idea of *crucial literal* to compute at most $n - 1$ *crucial literals* (i.e. conduct 1 experiments) to pairwise discriminate the theories. The incremental discrimination strategy we adopt will reduce the total space of possible theories, and consequently requires fewer experiments. The use of the system is illustrated by considering an example of electronic circuit diagnosis (see Figures 24 & 25). Theorist is asked to explain the *low* voltage at the node o (i.e., an observation $node(o, low)$). It generates four possible explanations which are mutually incompatible. The obvious problem is to attempt to gather evidence to discriminate the preferred explanation. We could apply the method suggested in [163] and perform pairwise discrimination, but this requires at least three experiments. In our incremental approach, the system incrementally computes *crucial literals* for a pair of theories and asserts the result supplied by the user. In this example, it needs to compute only two *crucial literals* to find the best of four theories, because the knowledge acquired during first two experiments has eliminated one of the subsequently constructed explanations.

Multiple explanations arise because of incomplete knowledge about the world being considered. The process of discriminating theories by identifying crucial literals and performing experiments is just a kind of knowledge growth that results in a smaller set of potential theories.

Theorist:
| ?- setof(D,explain([node(o,low)],D), X),
| length(X,L).
D = _0
X = [
[*not_ok*(*g*3), *not_ok*(*g*2), *not_ok*(*g*1)],
[*not_ok*(*g*3), *not_ok*(*g*2), *ok*(*g*1)],
[*not_ok*(*g*3), *ok*(*g*2), *ok*(*g*1)],
[*ok*(*g*3), *ok*(*g*2), *not_ok*(*g*1)]
]
L = 4
Extended Theorist:
[*theorist*]: explain node(o,low).
Experiment to test:node(f,low)?
|: yes.
Results of Experiment to be Asserted:node(f,low).
Experiment to test:node(e,high)?
|: no.
Results of Experiment to be Asserted:node(e,low).
Answer is: [*node*(*o*, *low*)]
Best Theory is: [*ok*(*g*3), *ok*(*g*2), *not_ok*(*g*1)]

Figure 25: Circuit Diagnosis and Incremental Theory Selection

Using this technique, we attempt to solve two serious problems. The first is the "multiple theories" problem of how to select the best theory. The second is to control the "combinatorial explosion of theories" by incrementally asserting the new facts.

Nevertheless, the enhanced Theorist lacks computational efficiency and fails to deal with a general problem of selecting a better theory. Since it is built on top of Theorist, the computation of the *crucial literal* between two theories is as expensive as the construction of the theories. The Seki and Takeuchi method requires a derivation structure similar to a derivation tree in Theorist, therefore, the overall computation performed by the enhanced Theorist framework is almost two times as expensive than the ordinary con-

struction of theories by Theorist. Further, the Seki and Takeuchi algorithm computes only one crucial literal between two incompatible theories. If the user fails to provide an answer (i.e., no conclusive result of the experiment), there is no way to find alternate experiment which may discriminate the theories, even if such experiment exists. In the next section we propose another extension which resolves these problems.

7.3 Identifying Crucial Literals

Rather than using the method suggested in [163] to compute a *crucial literal*, we examine the derivation tree generated by THEORIST while checking the consistency. We extend the consistency mechanism to form a consistency tree as described in Def. 3.

Suppose $T_1 = \{t_1\}$ and $T_2 = \{t_2\}$ are two theories, where both t_1 and t_2 are consistent. Therefore, $Facts \not\vdash \neg t_1$ and $Facts \not\vdash \neg t_2$. Further, we suppose that the consistency trees for $\neg t_1$ and $\neg t_2$ are as shown in Figure 26. Then any one of the four consistency trees identifies C as a crucial literal between T_1 and T_2. In other words, determining the truth value of C would refute one of T_1 or T_2.

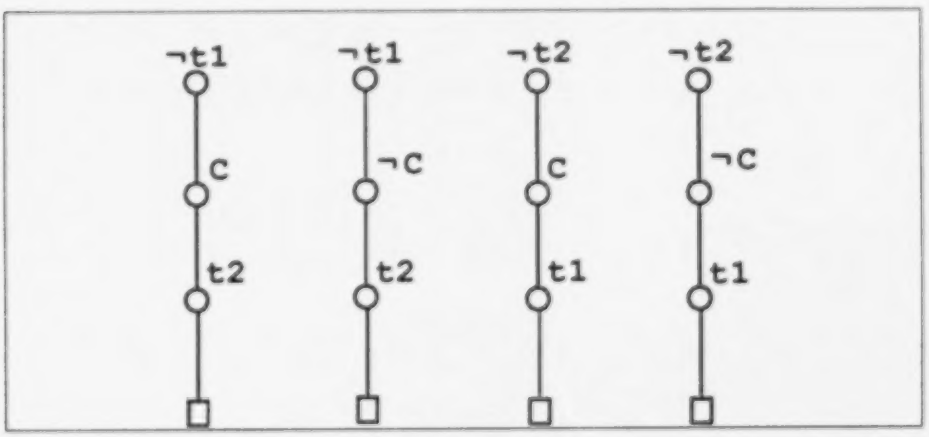


Figure 26: Consistency trees for t_1 and t_2

To explain an observation o, Theorist has to explain two subgoals a and b. If a is selected first, its proof would require hypothesis instance t_1. Before us-

fact $o \leftarrow a \wedge b$.
fact $o \leftarrow \neg a \wedge f$.
fact f.
fact $a \leftarrow f \wedge t_1$.
  fact $\neg a \leftarrow b \wedge t_2$.
  fact $b \leftarrow f \wedge t_3$.
  hypothesis t_1.
  hypothesis t_2.
  hypothesis t_3.

Example 7.2: THEORIST Database

ing t_1 in the proof tree, the theorem prover has to establish the unprovability of $\neg t_1$. If t_1 is consistent, then there is no success branch of the derivation tree constructed from the facts and $\neg t_1$. Figure 27 (a) shows one of the finitely failed branches. However, if we use new hypothesis instances to extend the failure branches, some of them become success -consistency trees. Figure 27 (b) shows a consistency tree for t_1 as defined in Def. 3. Figure 27 (c) shows a proof tree which is logically equivalent to the tree shown in Figure 27 (b). Each of its nodes represents an atomic goal.

Figure 27 (c) shows that $F \models t_3 \wedge t_2 \rightarrow \neg t_1$ (i.e., $F \cup t_1 \wedge t_2 \wedge t_3 \vdash false$); $\{t_1, t_2, t_3\}$ is an inconsistent set of hypothesis instances, or a nogood. Since facts are true in the world (i.e., $\models F$) the logical relationship $f \wedge t_3 \rightarrow b$ in Figure 27 (c) implies $t_3 \rightarrow b$ (using the deduction theorem). The following logical relationships may be obtained from $tree(\neg t_1)$ (see Figure 27 (c)):

$$t_3 \rightarrow b \tag{7.3}$$

$$b \wedge t_2 \rightarrow \neg a \rightarrow \neg t_1 \tag{7.4}$$

$$t_1 \wedge t_2 \rightarrow \neg b \tag{7.5}$$

$$t_2 \wedge t_3 \rightarrow \neg a \tag{7.6}$$

$$t_1 \rightarrow a \tag{7.7}$$

From formulas (7.3) and (7.5), an instance of possible hypothesis t_3 implies a literal b and instances t_1 and t_2 implies a literal $\neg b$. Therefore, the

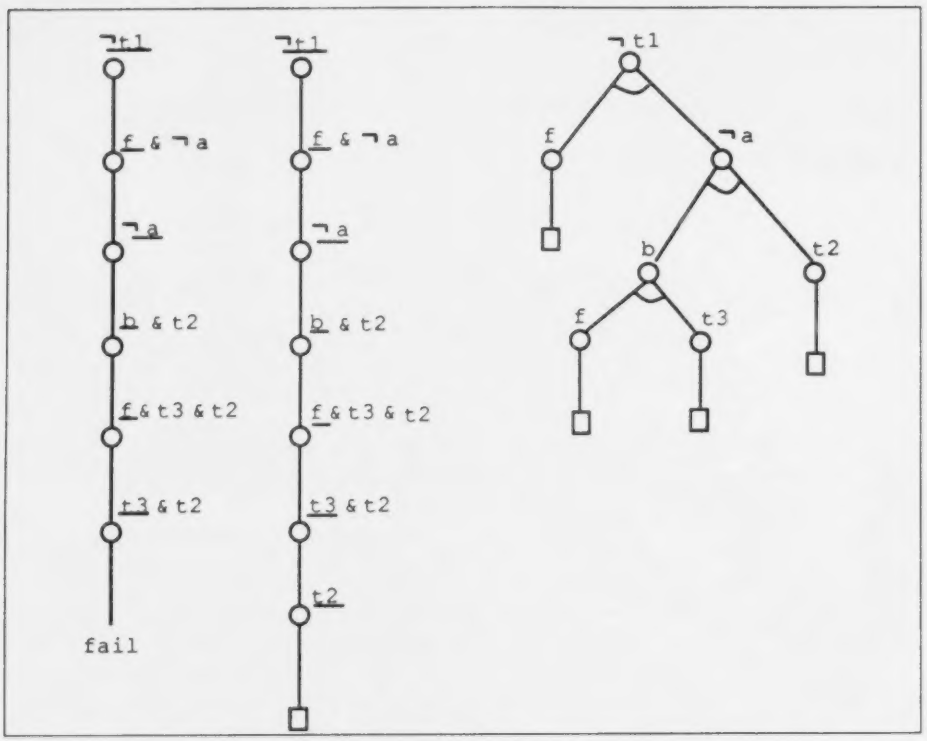


Figure 27: Consistency tree $tree(\neg t_1)$

literal b is a *crucial literal* between any two theories which contain the instance t_3 and the instances t_1 and t_2 respectively. Similarly we can show from formulas (7.6) and (7.7) that a is a *crucial literal* between two possible theories. Entries in Table 1 summarizes this information.

We now define the role of a *crucial literal* in a consistency tree, with the following theorem:

Theorem 42 *Suppose two theories T_1 and T_2 explain an observation O. Let C be an atomic internal node in a consistency tree, say $tree(\neg t_i)$, for some hypothesis instance $t_i \in T_1$ or T_2; let H_{st} be a non-empty set of hypothesis instances contained in the subtree $tree(C)$ and H_{ot} be a set of hypothesis instances contained in the consistency tree $tree(\neg t_i)$ including the root (i.e., hypothesis instance t_i) and excluding the subtree $tree(C)$; and there is no*

Set of hypotheses H_i	*Set of hypotheses* H_j	*Crucial Literal Between Theories:* T_i and T_j ($H_i \subseteq T_i$ and $H_j \subseteq T_j$)
$[t_3]$	$[t_1, t_2]$	b
$[t_3, t_2]$	$[t_1]$	$\neg a$

Table 7.4: Crucial literals and respective sets of hypothesis instances.

literal, say L, in the subtree $tree(C)$ such that $\neg L$ is an ancestor of C. Then C is a crucial literal between the theories T_1 and T_2 iff $H_{st} \subseteq T_1$ (or T_2) and correspondingly $H_{ot} \subseteq T_2$ (or T_1).

Proof: If there is a crucial literal between theories T_1 and T_2, then there exists a consistency tree which contains that literal (by definition), say $tree(\neg t_i)$.

Suppose, C is a *crucial literal* between theories T_1 and T_2, then $F \cup T_1 \models C$ (or $\neg C$) and $F \cup T_2 \models \neg C$ (or C). From consistency tree $tree(t_i)$, $Facts \models H_{st} \rightarrow C$ and $Facts \models H_{ot} \rightarrow \neg C$. Therefore, all hypothesis instances in H_{st} must be contained in one of the two theories, and all the instances from H_{ot} must be contained in other theory.

Conversely, we assume that $H_{st} \subseteq T_1$ (or T_2) and $H_{ot} \subseteq T_2$ (or T_1). Since each theory is a conjunction of the instances of possible hypotheses, the theories are $T_1 = H_{st}$ (or H_{ot}) $\cup H_1$ and $T_2 = H_{ot}$ (or H_{st}) $\cup H_2$, where H_1 and H_2 are some consistent sets of hypotheses instances, possibly empty. From the consistency tree $tree(t_i)$, $Facts \models H_{st} \rightarrow C$ and $Facts \models H_{ot} \rightarrow \neg C$. Therefore, we can conclude that $Facts \cup T_1$ (or T_2) $\models C(or\ \neg C)$ and similarly, $Facts \cup T_2$ (or T_1) $\models \neg C(or\ C)$. That is, C is a crucial literal. ■

Corollary 43 *Suppose C is a crucial literal, and H_{st} and H_{ot} are sets of hypothesis instances as specified in the above theorem; $H_{st} \subseteq T_1$ and $H_{ot} \subseteq T_2$; and $H_{st} \subset H_{ot}$ (or $H_{ot} \subset H_{st}$). Then, the following are true:*

- *if C is true, then T_1 is selected and T_2 becomes inconsistent (or both T_1 and T_2 become inconsistent if $H_{ot} \subset H_{st}$).*
- *if C is false, then both T_1 and T_2 are eliminated as inconsistent theories (or T_2 is selected and T_1 becomes inconsistent if $H_{ot} \subset H_{st}$).*

7.4 Diagnosing Multiple Faults

Diagnosis is a problem in which one must explain the discrepancy between the observed and correct system behavior by assuming some component (possibly multiple components) of the system is functioning abnormally. A diagnostic reasoning system must deal with two issues concerning computational efficiency. The first is efficient search in a complex space for all possible diagnoses for a given set of observations about the faulty system. The second is efficient discrimination amongst multiple competing diagnoses. This section describes how crucial literals can be used to improve the empirical efficiency of diagnostic reasoning.

Since most diagnostic reasoning systems reason with incomplete knowledge, they must consider the alternate diagnoses. These alternate diagnoses, often leads to mutually incompatible explanatory theories. An extended system identifies possible decision points in the form of crucial literals during consistency checks of hypothesis instancea, used in the process of diagnosis construction. While exploring the combinatorial search space, each crucial literal provides an opportunity to perform an experiment. The result[1] of such experiment is then asserted as new knowledge which eliminates one of the two potential diagnoses containing the incorrect choice (or assumption). This new knowledge enables the system to identify new nogoods. All future diagnoses containing the resultant *nogood* are eliminated, which otherwise would be generated because of this incorrect choice. However, if the user fails to perform an experiment requested by the system, then the system provides another possible clue by identifying another crucial literal.

In what follows, we discuss three examples commonly used in diagnosis [57, 140, 29] to show the application of crucial literals to the general problem of diagnosing multiple faults and discriminating amongst competing diagnoses. The results presented here are from both basic Theorist and extended Theorist.

7.4.1 Medical Diagnosis

This example considers a simple problem of medical diagnosis which was used by Pople [140] to illustrate the use of branching logic for discriminating

[1] In our prototype system, it is the user's response to the query.

among the competing multiple diseases in a single patient. In our representation, we assume all diseases are equally plausible. However, this problem can be represented in a hierarchical structure similar to the representation for INTERNIST [140, page 184] as well. Here, we note that there are three mutually exclusive symptoms of edema and three mutually exclusive symptoms of abdominal pain. Each of these symptoms is caused by a different disease. The rules that describe how the diseases cause symptoms and general hypotheses are represented in the Theorist framework as shown in Example 7.3.

fact has_edema ← edema_symptom(X).
fact edema_symptom(proteinurea) ← not_ok(kidney).
fact edema_symptom(dysphnea) ← not_ok(heart).
fact edema_symptom(jaundice) ← not_ok(liver).
fact has_abdominal_pain ← abdominal_pain_symp(X).
fact abdominal_pain_symp(nausea) ← problem_is(indigestion).
fact abdominal_pain_symp(irritation_in_bowl) ← problem_is(dysentry).
fact abdominal_pain_symp(heart_burn) ← problem_is(acidity).
fact not edema_symptom(proteinurea) ← edema_symptom(dysphnea).
fact not edema_symptom(dysphnea) ← edema_symptom(jaundice).
fact not edema_symptom(jaundice) ← edema_symptom(proteinurea).
fact not abdominal_pain_symp(nausea) ←
abdominal_pain_symp(irritation_in_bowl).
fact not abdominal_pain_symp(irritation_in_bowl)←
abdominal_pain_symp(heart_burn).
fact not abdominal_pain_symp(heart_burn)←
abdominal_pain_symp(nausea).
hypothesis problem_is(X):problem_is(X).
hypothesis not_ok(X):not_ok(X).

Example 7.3: Medical Database

Figure 28 indicates that basic Theorist generates 9 possible potential diagnoses for the observations [*has_edema*, *has_abdominal_pain*]. By using the techniques proposed here, extended Theorist generates the best diagnosis

Theorist:
| ?- setof(X,explain([has_edema,has_abdominal_pain], X),D),
| length(D,L).
X = _0
D = [
[$problem_is(acidity), not_ok(heart)$],
[$problem_is(acidity), not_ok(kidney)$],
[$problem_is(acidity), not_ok(liver)$],
[$problem_is(dysentry), not_ok(heart)$],
[$problem_is(dysentry), not_ok(kidney)$],
[$problem_is(dysentry), not_ok(liver)$],
[$problem_is(indigestion), not_ok(heart)$],
[$problem_is(indigestion), not_ok(kidney)$],
[$problem_is(indigestion), not_ok(liver)$]
]
L = 9
Extended Theorist:
[*theorist*]: explain has_edema,has_abdominal_pain.
Experiment to test:edema_symptom(proteinurea)?
|: yes.
Results of Experiment to be Asserted:edema_symptom(proteinurea).
Experiment to test:not abdominal_pain_symp(nausea)?
|: no.
Results of Experiment to be Asserted:abdominal_pain_symp(nausea).
Answer is: [$has_edema, has_abdominal_pain$]
Best Theory is: [$problem_is(indigestion), not_ok(kidney)$]

Figure 28: Diagnoses generated by Theorist and Extended Theorist

by performing only two experiments and exploring the total space of three possible diagnoses.

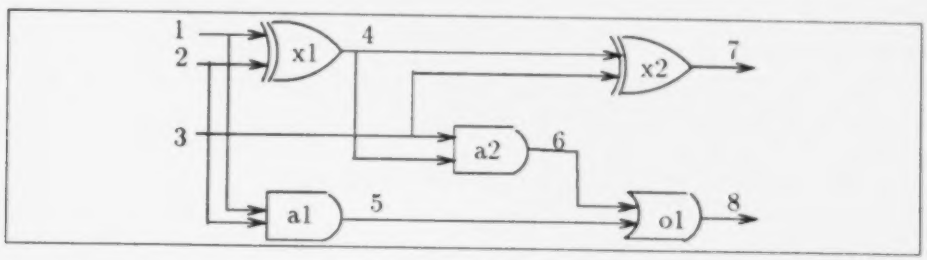


Figure 29: A full Adder

7.4.2 Full Adder Problem

Figure 29 shows a binary full adder consisting of five logical gates. This example is used extensively by Genesereth [57]. The circuit is represented in the Theorist framework as facts (how the logical gates normally behave) and possible hypotheses (typical failures). For the given input values at the nodes 1, 2 and 3, we ask the system to explain the output at the nodes 7 and 8. When we ask Theorist to explain the observation $node(7, high), node(8, low)$; it gives eight consistent explanations. Our system requires only three experiments to select a better diagnosis, as shown in Figure 30.

7.4.3 The Multiplier Problem

In this example, we consider the circuit shown in Figure 31. It consists three multipliers $m1$, $m2$ and $m3$, and two adders $a1$ and $a2$. In their paper, de Kleer and Williams [29] used this example to illustrate their *general diagnostic engine* (GDE). For the given inputs at the nodes a, b, c, d, e, and f, the system explains the output at the nodes j and k. The Theorist representation is given as in Example 7.5.

In addition to *facts* and *hypotheses*, the Theorist representation of this problem contains a third kind of statement called *meta*. *Meta* means that these statements are interpreted as simple Prolog statements by the Prolog-interpreter; this facilitates the use of Prolog built_in predicates in the Theorist representation.

```
hypothesis ok(G).                    fact and(a2).
hypothesis not_ok(G).                fact or(o1).
fact xor(x1).                        fact node(1,high).
fact xor(x2).                        fact node(2,low).
fact and(a1).                        fact node(3,high).
fact not node(X,low) ← node(X,high).
fact and_gate(G,high,high,high)← and(G), ok(G) .
fact and_gate(G,low,high,low)← and(G), ok(G) .
fact and_gate(G,high,low,low)← and(G), ok(G) .
fact and_gate(G,low,low,low)← and(G), ok(G) .
fact and_gate(G,high,high,low)← and(G), not_ok(G) .
fact and_gate(G,low,high,high)← and(G), not_ok(G) .
fact and_gate(G,high,low,high)← and(G), not_ok(G) .
fact and_gate(G,low,low,high)← and(G), not_ok(G) .
fact or_gate(G,high,high,high)← or(G), ok(G) .
fact or_gate(G,low,low,low)← or(G), ok(G) .
fact or_gate(G,low,high,high)← or(G), ok(G) .
fact or_gate(G,high,low,high)← or(G), ok(G) .
fact or_gate(G ,high,high,low)← or(G), not_ok(G) .
fact or_gate(G ,high,low,low)← or(G), not_ok(G) .
fact or_gate(G ,low,high,low)← or(G), not_ok(G) .
fact or_gate(G ,low,low,high)← or(G), not_ok(G) .
fact xor_gate(G,high,high,low)← xor(G), ok(G) .
fact xor_gate(G,low,high,high)← xor(G), ok(G) .
fact xor_gate(G,high,low,high)← xor(G), ok(G) .
fact xor_gate(G,low,low,low)← xor(G), ok(G) .
fact xor_gate(G ,high,high,high)← xor(G), not_ok(G) .
fact xor_gate(G ,high,low,low)← xor(G), not_ok(G) .
fact xor_gate(G ,low,high,low)← xor(G), not_ok(G) .
fact xor_gate(G ,low,low,high)← xor(G), not_ok(G) .
fact node(4,O)← node(1,E), node(2,F), xor_gate(x1,E,F,O).
fact node(7,O)← node(4,E), node(3,F), xor_gate(x2,E,F,O).
fact node(5,E)← node(3,A), node(4,B), and_gate(a2,A,B,E).
fact node(6,F)← node(1,C), node(2,D), and_gate(a1,C,D,F).
fact node(8,O)← node(5,E), node(6,F), or_gate(o1,E,F,O).
```

Example 7.4: Theorist Database for a Full Adder

Theorist:

| ?- setof(X,explain([node(7,high),node(8,low)], X),D),
| length(D,L).

X = _0

D = [

[$not_ok(o1), not_ok(a1), not_ok(a2), not_ok(x2), ok(x1)$],
[$not_ok(o1), not_ok(a1), not_ok(a2), ok(x2), not_ok(x1)$],
[$not_ok(o1), not_ok(a1), ok(a2), not_ok(x2), ok(x1)$],
[$not_ok(o1), not_ok(a1), ok(a2), ok(x2), not_ok(x1)$],
[$not_ok(o1), ok(a1), not_ok(a2), ok(x2), not_ok(x1)$],
[$not_ok(o1), ok(a1), ok(a2), not_ok(x2), ok(x1)$],
[$ok(o1), ok(a1), not_ok(a2), not_ok(x2), ok(x1)$],
[$ok(o1), ok(a1), ok(a2), ok(x2), not_ok(x1)$]

]

L = 8

Extended Theorist:

[*theorist*]: explain node(7,low),node(8,high).
Experiment to test:node(6,low)?
|: yes.
Results of Experiment to be Asserted:node(6,low).
Experiment to test:node(5,high)?
|: no.
Results of Experiment to be Asserted:node(5,low).
Experiment to test:node(4,high)?
|: yes.
Results of Experiment to be Asserted:node(4,high).
Answer is: [$node(7, high), node(8, low)$]
Best Theory is: [$ok(o1), ok(a1), not_ok(a2), not_ok(x2), ok(x1)$]

Figure 30: Explanations generated by Theorist and Extended Theorist

```
fact m(m1).
fact m(m2).
fact m(m3).
fact a(a1).
fact a(a2).
fact node(a,3).
fact node(b,2).
fact node(c,3).
fact node(d,2).
fact node(e,3).
fact node(f,2).
fact multiplier(M,X,Y,Z)←m(M),mult(X,Y,Z),ok(M).
fact multiplier(M,X,Y,Z)←m(M),mult(X,Y,Z1),ne(Z1,Z),
not_ok(M).
fact adder(M,X,Y,Z)←a(M),plus(X,Y,Z),ok(M).
fact adder(M,X,Y,Z)←a(M),plus(X,Y,Z1),ne(Z1,Z),not_ok(M).
fact node(g,G)← node(a,A),node(b,B),multiplier(m1,A,B,G).
fact node(h,H)← node(c,C),node(d,D),multiplier(m2,C,D,H).
fact node(i,I)← node(e,E),node(f,F),multiplier(m3,E,F,I).
fact node(j,J)← node(h,H),node(g,G),adder(a1,G,H,J).
fact node(k,K)← node(h,H),node(i,I),adder(a2,H,I,K).
fact not node(X,Y)← ne(X,Y,Z),node(X,Z).
hypothesis ok(G).
hypothesis not_ok(G).
meta plus(X,Y,Z).
meta minus(X,Y,Z).
meta mult(X,Y,Z).
meta ne(X,Y).
meta ne(X,Y,Z).
```

Example 7.5: Theorist Database for the Multiplier Problem

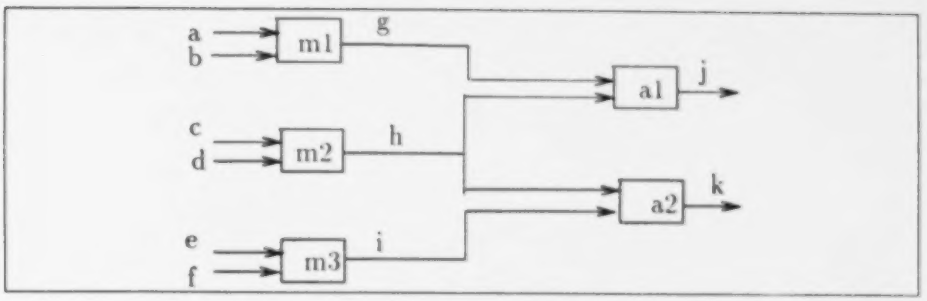


Figure 31: A Multiplier

There are two axioms which describe the function of multipliers and adders. The first indicates the correct functioning of the device. The second represents the faulty device. Since devices like multipliers and adders are made of many small subcomponents, it is unrealistic to assume that the device is completely broken. Therefore, faulty device means one or more subcomponents are defective and lead to incorrect output. There is an exponentially large number of hypothetical possibilities for incorrect outputs. However, for the sake of simplicity and clear illustration of the approach taken in this paper, we assume that the incorrect outputs differ from correct output by 2. Considering the same input and the output as in [29], our system needs three experiments to select the best theory, as shown in Figure 32.

7.5 Conclusions and Future Work

Selecting a better theory is one of the main issues of scientific theory formation. In this chapter, we used Popper's criteria for selecting better theories by performing crucial experiments. The idea of *crucial literal* [163] is the basis for identifying these experiments.

The Seki and Takeuchi [163] method computes a *crucial literal* between two incompatible Horn clause theories, and the truth value of this literal discriminates a better theory. However, their method is based on a notion of integrity constraint and says nothing about identifying alternative crucial literals when the user fails to determine the truth value of the first crucial literal. The following is a summary of our contributions of this chapter:

Theorist:

| ?- setof(D,explain([node(j,10),node(k,12)],D),X),
| length(X,L).

D = _0

X = [

$[not_ok(a2), not_ok(m3), not_ok(a1), ok(m1), ok(m2)]$,
$[not_ok(a2), not_ok(m3), ok(a1), not_ok(m1), ok(m2)]$,
$[not_ok(a2), not_ok(m3), ok(a1), ok(m1), not_ok(m2)]$,
$[not_ok(a2), ok(m3), ok(a1), ok(m1), not_ok(m2)]$,
$[ok(a2), not_ok(m3), ok(a1), ok(m1), not_ok(m2)]$,
$[ok(a2), ok(m3), not_ok(a1), ok(m1), ok(m2)]$,
$[ok(a2), ok(m3), ok(a1), not_ok(m1), ok(m2)]$

]

L = 7

Extended Theorist:

[*theorist*]: explain node(j,10),node(k,12).
Experiment to test:node(i,6)?
|: yes.
Results of Experiment to be Asserted:node(i,6).
Experiment to test:node(g,6)?
|: yes.
Results of Experiment to be Asserted:node(g,6).
Experiment to test:node(h,6)?
|: no.
Results of Experiment to be Asserted:node(h,4).
Answer is: $[node(j, 10), node(k, 12)]$
Theory is: $[not_ok(a2), ok(m3), ok(a1), ok(m1), not_ok(m2)]$

Figure 32: Explanations generated by Theorist and Extended Theorist

- First, we adapt Seki and Takeuchi's method for Theorist without using the notion of integrity concept. To do so, we incrementally use *crucial literals* as experiments to reduce the total number of explanations generated for a given set of observations, by asserting the results of experiments as new knowledge. Then, we specify an extension which incrementally constructs a table for all possible crucial literals for any pair of theories. This extension is relatively more efficient than the simple adoption of Seki and Takeuchi's method, and provides the user with greater opportunity to conduct experiments to eliminate *falsifiable* theories.

- As a diagnostic reasoning system, our framework provides a uniform treatment to the problems of characterizing the space of possible diagnoses and distinguishing among competing diagnoses.

- Unlike Seki and Takeuchi [163] and Shapiro [167], we deal with first-order logic theories and compute crucial literals without using the notion of integrity constraint. The number of experiments (or tests) is typically much fewer than what is required in these other approaches.

Our approach is based on scientifically investigating clues for discriminating among competing theories. However, it is not always possible to perform all experiments successfully. In such cases, the tendency is to use some kind of statistically founded knowledge (i.e., past experience) or some simplicity criteria in selecting among the multiple competing theories. In order to achieve a scientifically more valid and practically efficient reasoning system, we believe we should first attempt to use the most rational and scientific criteria (i.e., performing experiments), then use measures based on biases (e.g. simplicity, most specific knowledge, probability, persistence, etc.). Since the computational complexity of clausal logic-based reasoning systems is beyond asymptotic analysis, we can only talk about empirical efficiency by considering some standard examples.

Chapter 8

Conclusion

8.1 Summary of Contributions

The major achievement of this dissertation has been a general problem solving paradigm that is both representationally expressive and empirically efficient and which generalizes and integrates several distinct reasoning approaches in a simple and intuitive way.

The proposed framework provides a new approach to truth maintenance systems based on consistency trees, which turns out to be a general paradigm to unify other useful empirical techniques such as meta-reasoning and constraint processing. Our approach overcomes many limitations of the existing systems as regards their expressive power and computational efficiency. Up to now, these techniques have been studied under separate research sub-areas of AI. This research will help to resolve apparent differences and distinctions in these sub-areas.

A rational approach based on crucial literals (cf., scientific experiments) is proposed to incrementally distinguish between incompatible competing theories, which, in turn, makes the reasoning more efficient. Its application in diagnostic reasoning is explored. Further, the same idea is used to provide a unified treatment of theory selection combining several other approaches.

While reasoning with incomplete knowledge, the problem-solver has two main considerations: efficiently searching for a solution and reducing the total number of potential solutions (i.e., multiple extensions) which may be exponentially large. Our approach exploits advantages of constraint processing

and meta-reasoning along with truth maintenance to efficiently find a solution; and reduces the total number of solutions by incrementally compiling inconsistencies and eliminating refuted theories.

A complete implementation of a prototype hypothetical reasoning system based on the techniques developed in this thesis is yet to be realized. Consequently, we face the fundamental question about how to compare empirical efficiency of our framework with other reasoning systems. While we partly agree with the criticism that the performance evaluation of the proposed reasoning framework cannot be made on large empirical data without a complete implementation, we argue that the partial implementation and the already established results in the area of constraint satisfaction provide sufficient evidence to conclude that our approach is empirically efficient. We implemented the idea of incremental theory selection using crucial literals which demonstrated how an exponentially large explanation space can be reduced. We have shown that consistency tree based pruning of inconsistent search space is equivalent to forward checking. Thus, the experimental and analytical result [67] (forward checking is the best strategy in comparison to other techniques with respect to the cost of preprocessing and the number of backtracks) shows that our approach is better than the ATMS style of reasoning. Also, a dry run of the queens problem further demonstrated the advantages of our approach.

8.2 Future Research

There are many directions both practical and theoretical in which this research can be extended. The following are the main issues that need to be addressed in the future:

A complete implementation

Our first task would be to extend the partial implementation to a complete and stable prototype working system using Quintus Prolog on Sun-4 workstations. The basis for this implementation should be XTheorist system: a hypothetical reasoning framework developed at the University of Alberta. The XTheorist architecture consists of three layers: the window interface layer, the inference engine and database layer. Basically, we have to inte-

grate our partial implementation[1] with the inference engine and extend it to construct truth tables and then take advantage of incrementally accumulated information. The other two layers provide a nice interface to interact with XTheorist.

The complete implementation of our system would be an expressively rich and computationally efficient problem solving system. It will enable us to experiment with potential specifications of the reasoning problems and with various domain-dependent heuristics for solving problems such as constraint satisfaction problems.

A constraint logic programming framework

The integration of constraint satisfaction paradigm within the logic programming languages has been an active research topic in logic programming and AI around the world for the last few years (e.g., CHIP [179, 180, 181], constraint logic programming scheme (CLP) [81], Prolog III [18], Trilogy [182], CAL [2], Echidna [69]). The main motivation for developing all of these Horn clause logic-based constraint solving systems is to actively use constraints to reduce the unnecessary search space in *a priori* way, in contrast to the standard backtracking which prune the space in *a posteriori* way (i.e., after having discovered a failure).

Undoubtedly, these constraint programming systems have a great advantage over the the standard Prolog systems. However, these systems lack expressive richness (i.e., Horn clause vs. full clause and meta-level vs. object-level specification) and cannot be used in nonmonotonic reasoning environment. Constraint processing alone is not enough to achieve efficiency, we also need truth maintenance component in our reasoning system.

The ideas presented in Chapter 5 could be extended for developing a more general logic programming system by introducing the notion of hypothesis that will overcome the above limitations. There are plenty of avenues to study the theoretical aspects as well as the practical issues involved in such a system.

---

[1] We have implemented the idea of consistency tree and crucial literals in Cprolog.

Unified Theory Preference

Several formalisms have been proposed to capture the rational inference typically involved in common sense reasoning—for instance, minimal model approaches (e.g., [113, 115]) and consistency approaches (e.g., [145, 119, 125, 136, 134]). Essentially, these frameworks reason with incomplete knowledge (knowledge with exceptions, tentative knowledge, etc.) which often results in multiple competing solutions for a given problem—the so called multiple extension problem [41, 66]. Unfortunately, the number of multiple competing theories may be exponentially large (e.g., for n components, a diagnostic reasoning system may generate 2^n possible diagnoses).

Most of the above proposals attempt to choose between or eliminate competing theories based on domain-dependent knowledge. Many seemingly disparate means of theory selection have been suggested, for instance, using askables [131]; crucial literals [153]; specificity [137]; probability [126, 5, 62]; persistence and chronological ordering in temporal reasoning [168, 83, 96, 60]; priorities among defaults [12]; fixed predicates [115, 132], etc.

In [161], we have described how the consistency tree can be used to unify theory selection/elimination approaches based on *crucial literals*, *askables*, *testables* and *fixed predicates*. We could extend this in two directions: first, investigating how other theory preference and pruning criteria such as *simplicity*, *specificity*, *priorities*, *probabilities* etc. can be dealt with using the consistency tree. These criteria can be applied over the pairs of *insets* and *outsets* (viewed as pairs of incompatible sub-theories) which in turn selects and eliminates theories at the sub-theory level. Further, research is required to identify under what conditions incremental elimination of theories at the sub-theory level is possible.

The idea of crucial literals is based on scientifically investigating clues for discriminating among competing theories. However, it is not always possible to perform all experiments successfully. In such cases, the tendency is to use some kind of statistically founded knowledge (i.e., past experience) or some simplicity criteria in selecting among the multiple competing theories. In order to achieve a scientifically more valid and practically efficient reasoning system, we should first attempt to use the most rational and scientific criteria (i.e., performing experiments if affordable), then use measures based on biases (e.g, simplicity, most specific knowledge, probability, persistence, etc.). The framework developed in this dissertation could be extended to have a partial

order among the various preference criteria.

Logic-based framework for learning

This dissertation is based on two fundamental ideas: incrementally accumulating information from the reasoning environment and enabling the reasoning agent to perform experiment in the world to grow its knowledge such that the better conclusions can be obtained. The framework for truth maintenance in hypothetical reasoning, as described in chapter 4, could be extended as a logic-based framework for learning. [2] For example, the problem "Generalisation as search" [122] could be formulated from the perspective of hypothetical reasoning. The *consistency tree* (see Def. 3) will be a key in identifying potentially learnable formulas and studying their properties. We believe that formalizing learning under the framework of extended-Theorist system would enable us to investigate formal properties of learning and to verify them by experimenting with the working hypothetical reasoning system. Similar attempts have been made in [78] for understanding learning from the perspective of hypothetical reasoning.

Rational Belief Revision

One of the fundamental aspects of intelligent behavior is the revision of beliefs in the view of new information. Beliefs not merely provide a basis for the reasoning process, but also are valuable and quite expensive to obtain. The extended-THEORIST framework could be developed as a rational belief revision framework.

The intuition is that if the new information logically contradicts the original (consistent) set of beliefs, then its effects are three-fold: a subset of the original belief set not effected at all; a subset of the original beliefs is falsified by the new information; and the rest of the beliefs which were originally true become tentative (see Figure 33). While there is no uncertainty about retaining the set of unaffected beliefs and rejecting the falsified beliefs, the third category introduces ambiguity that leads to more than one possible way of revising beliefs. Most of the existing theories of rational belief revision (e.g.,

[2] THEORIST has already been shown to be a useful tool for the specification of analogical reasoning [185, 58].

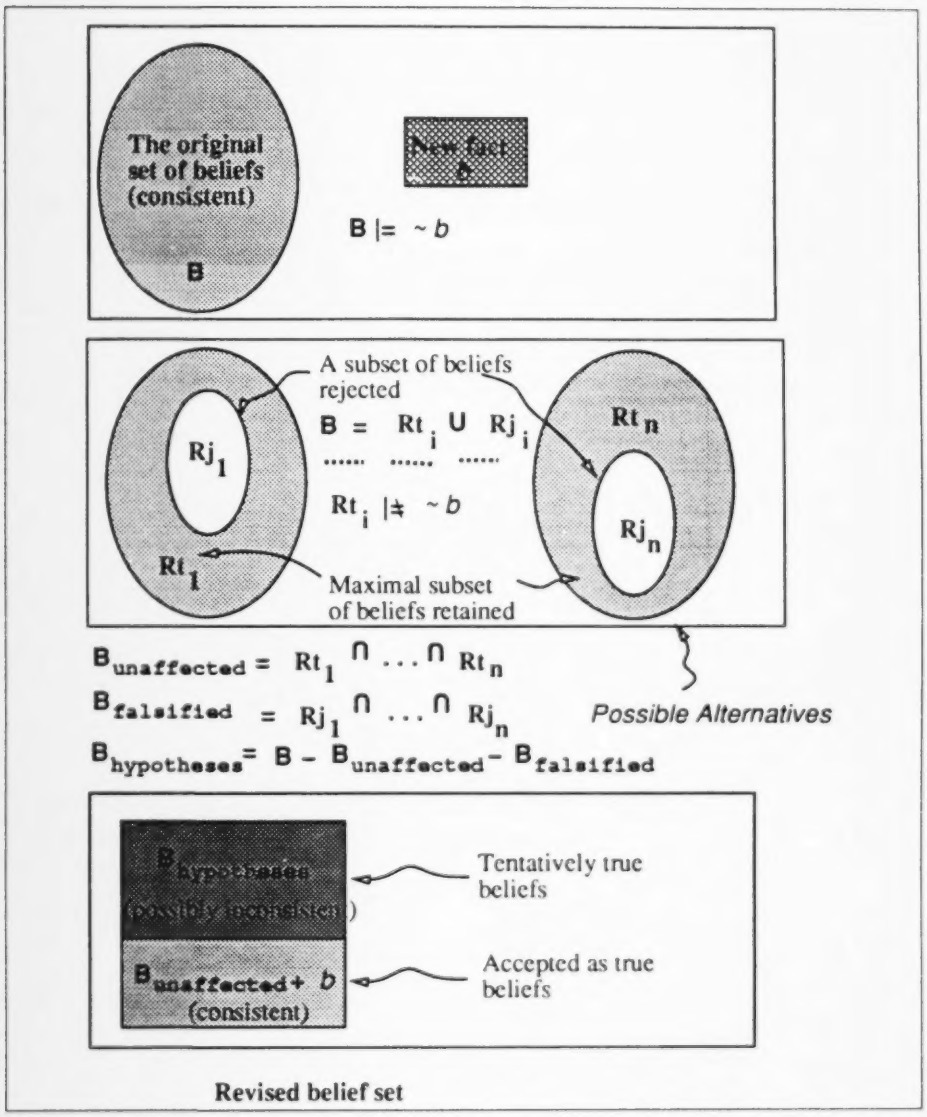


Figure 33: Revising Beliefs as Hypotheses

[104, 3, 52]) attempt to minimize the possible multiple revisions by assigning some sort of partial order among the beliefs. Then an intersection is taken of the remaining revisions to identify the set of beliefs consistent with the new information. Consequently, we are not only losing many valuable beliefs from the third category, but also changing the truth status of some of the beliefs from doubtful to true, that might turn out to be counterintuitive.

The beliefs, whose truth status have changed to doubtful, are very useful in the real world, and should not be discarded just to get a unique revision. Rather, the beliefs should be maintained in terms of two epistemic classes: those accepted as true—facts (consistent) and those accepted as potentially refutable—hypotheses (possibly inconsistent). Then each revision is the union of facts and a maximal consistent subset of hypotheses. Based on this classification we could develop a belief revision framework which allows possible multiple revisions and offers more flexible ways of choosing among alternatives. For example, *crucial literals* would be a rational criteria to select among multiple beliefs. We could argue that rather than throwing away many useful beliefs, it is better to acquire new belief (result of crucial test) which might select the better revisions. Other criteria of theory selection can be integrated in a simple way. Also, this framework would unify the non-monotonic reasoning and belief revision (see [152] for research in progress on these ideas).

Potential Applications

Since the hypothetical reasoning framework presented in this thesis unifies a number of heuristic-based techniques from different sub-areas of AI for efficiently performing the reasoning tasks, it may be applied in a number of problem domains. For example, we could develop a medical diagnosis system for a non-trivial domain of the real world. Computer vision, planning and qualitative reasoning are the other important areas where we can explore potential applications of this framework. For example, resolving ambiguities among the multiple plans using the idea of crucial literals and solving the task of plan formation efficiently by integrating the constraint processing with truth maintenance.

Our recent work [159] has introduced the idea of second-level consistency using consistency trees to reduce the explanation space for circumscriptive theorem provers.

Bibliography

[1] W. Ackermann. *Solvable cases of the Decision problem.* North-Holland, Amsterdam, The Netherlands, 1954.

[2] A. Aiba, K. Sakai, Y. Sato, D. J. Hawley, and R. Hasegawa. Constraint logic programming language CAL. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 263–276, Tokyo, Japan, November 28-December 2 1988. ICOT.

[3] C. E. Alchourron, P. Gardenfors, and D. Makinson. On the logic logic of theory change: partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510–530, 1985.

[4] A. R. Anderson and N. D. Belnap. *Entailment: The Logic of Relevance and Necessity*, volume 1. Princeton University Press, Princeton, NJ, 1975.

[5] F. Bacchus. Statistically founded degrees of belief. In *Proceedings of the Seventh Bienniel Conference of the CSCSI*, pages 59–66, Edmonton, Alberta, June 6-10 1988.

[6] H. Bacha. Meta-level programming: a compiled approach. In *Proceedings of the Fourth International Conference on Logic Programming*, volume 1, pages 394–410, Melbourne, Australia, May 1987.

[7] W. Bibel. Constraint satisfaction from a deductive viewpoint. *Artificial Intelligence*, 35(3):401–413, July 1988.

[8] K. A. Bowen. Meta-level programming and knowledge representation. *New Generation Computing*, 3(3):359–383, 1985.

[9] K. A. Bowen and R. A. Kowalski. Amalgamating language and metalanguage in logic programming. In K.L. Clark and S-Å. Tärnlund, editors, *Logic Programming*, pages 153–172. Academic Press, New York, 1982.

[10] R. J. Brachman and B. C. Smith (ed.). *Special Issue on Knowledge Representation*. Number No. 70. SIGART Newsletter, February 1980.

[11] R. J. Brachman and H. J. Levesque (ed.). *Readings in Knowledge Representation*. Morgan Kaufmann Publishers, Inc., California, 1985.

[12] G. Brewka. Preferred subtheories: An extended logical framework for default reasoning. In *Proceedings of Eleventh International Joint Conference on Artificial Intelligence, IJCAI-89*, pages 1043–1048, Detroit, USA, August 1989.

[13] Gerhard Brewka. *Nonmonotonic Reasoning: From Theoretical Foundations Towards Efficient Computation*. PhD thesis, Fachbereich Informatik der Universitat Hamburg, May 1989.

[14] E. Charniak and D. McDermott. *Introduction to Artificial Intelligence*. Addision-Wesley, 1985.

[15] W. Clancy and C. Bock. MRS/NEOMYCIN: Representing metacontrol in predicate calculus. Report No. HPP-82-3, Stanford Heuristic Programming Project, Stanford, 1982.

[16] L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[17] A. Colemerauer. Prolog II: Mannuel de reference et modele theorique. Technical Report GIA, Faculte de Sciences de Luminy, March 1982.

[18] A. Colemerauer. Opening Prolog III universe. *BYTE Magazine*, 12(9), August 1987.

[19] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Ann. ACM Symp. on Theory of Computing*, pages 151–158, New York, 1971.

[20] R. Corlett, N. Davies, R. Khan, H. Reichgelt, and F. van Harmelen. Socrates: A flexible toolkit for building logic based expert systems. In P. Jackson, H. Reichgelt, and F. van Harmelen, editors, *Logic-based Knowledge Representation*. MIT Press, 1988.

[21] P. T. Cox. Finding backtrack points for intelligent backtracking. In J. A. Campell, editor, *Implementation of Prolog*, pages 194–215. Ellis Harwood, 1984.

[22] Nigel Cutland. *Computability: an introduction to recursive function theory*. Cambridge University Press, first edition, 1980.

[23] L. S. Davis and A. Rosenfield. Cooperative processes for low level vision: a survey. *Artificial Intelligence*, 17:245–263, 1981.

[24] R. Davis. Meta-rules: Reasoning about control. *Artificial Intelligence*, 15:179–222, 1980.

[25] J. de Kleer. Choices without backtracking. In *Proceedings of AAAI-84*, pages 79–85, Austin, TX, 1984.

[26] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[27] J. de Kleer. A comparison of ATMS and CSP techniques. In *Proceedings of Eleventh International Joint Conference on Artificial Intelligence*, pages 290–296, Detroit, USA, August 1989.

[28] J. de Kleer, J. Doyle, and G. Sussman. AMORD: Explicit Control of Reasoning. In *Proceedings of Symposium on Artificial Intelligence and Programming Languages. SIGPLAN Notices*, volume 12 (8), pages 116–132, August 1977.

[29] J. de Kleer and B.C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[30] D. De Schreye and M. Bruynooghe. The compilation of forward checking regimes through meta-interpretation and transformation. In J. W. Lloyd, editor, *Proceedings of the workshop on Meta-Programming in Logic programming*. University of Bristol, Bristol, June 1988.

[31] R. Dechter. Learning while searching in constraint satisfaction problems. In *Proceedings of the AAAI-86 Conference*, pages 178–183, Philadelphia, Pennsylvania, August 11-15 1986. University of Pennsylvania.

[32] J. P. Delgrande. A first-order conditional logic for prototypical properties. *Artificial Intelligence*, 33(1):105–130, 1987.

[33] J. P. Delgrande. An approach to default reasoning based on a first-order conditional logic: Revised report. *Artificial Intelligence*, 36:63–90, 1988.

[34] M. Dincbas, H. Simonis, and P. Van Hentenryck. Extending equation solving and constraint handling in logic programming. In *Colloquium on Resolution of equations in Algabraic structures (CREAS)*, Texas, May 1987.

[35] W. Dowling and J. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, (3):267–284, 1984.

[36] J. Doyle. Truth maintenance systems for problem solving. Technical Report AI-TR-419, MIT, AI Lab, Cambridge, MA, 1978.

[37] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[38] O. Dressler. An extended basic atms. In M. Reinfrank, J. de Kleer, and M.L. Ginsberg, editors, *Non-Monotonic Reasoning*, number 346 in Lecture Notes in Artificial Intelligence, pages 143–163. Springer Verlag, Berlin, 1989.

[39] David W. Etherington. *Reasoning with Incomplete Information.* Pitman (London)/ Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.

[40] D.W. Etherington, A. Borgida, R. J. Brachman, and H. Kautz. Vivid knowledge and tractable reasoning: Preliminary report. In *Proceedings of AAAI-90*, pages 1146–1152, 1990.

[41] D.W. Etherington and R. Reiter. On inheritance hierarchies with exceptions. In *Proceedings of AAAI-83*, pages 104–108, 1983.

[42] G. Ferguson. Identity and skolem functions in resolution-based hypothetical reasoning. Master's thesis, Department of Computing Science, University of Alberta, August 1989.

[43] R. E. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[44] K. D. Forbus and J. de Kleer. *Building Problem Solvers: Program notes on Truth Maintenance Systems.* 1990. [Draft prepared for AAAI-90 tutorial on Truth Maintenance Systems].

[45] E. C. Freuder. Synthesizing constraint expressions. *Comm. ACM*, 21:958–966, 1978.

[46] E. C. Freuder. A sufficient condition for backtrack free search. *J. ACM*, 29(1):24–32, 1982.

[47] A. M. Frisch. Using model theory to specify AI programs. In *Proceedings of the IJCAI-85*, pages 148–154, Los Angels, CA, 1985.

[48] J.D.D. Gagné. Perceptual reasoning: A logicl foundation for computer vision. Research report CS-87-67, Department of Computer Science, University of Waterloo, Waterloo, Ontario, December 1987.

[49] H. Gallaire. Logic programming: further developments. In *Proceedings of IEEE Symposium on Logic Programming*, pages 85–99, Boston, July 1985.

[50] M. Gallaire and C. Lasserre. Controlling knowledge deduction in a declarative approach. In *Proceedings of the IJCAI-79*, pages S1–S6, Tokyo, Japan, 1979.

[51] M. Gallaire and C. Lasserre. Meta-level control for logic programs. In K.L. Clark and S-Å. Tärnlund, editors, *Logic Programming*, pages 173–178. Academic Press, New York, 1982.

[52] Peter Gardenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States.* MIT Press, Cambridge, MA, 1988.

[53] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness.* W. H. Freeman and Company, San Francisco, 1979.

[54] J. Gasching. A problem similarity approach to devising heuristic: first results. In *Proceedings of IJCAI-79*, pages 301–307, Tokyo, Japan, 1979.

[55] M. Genesereth and D. Smith. An overview of meta-level architecture. In *Proceedings of AAAI-83*, pages 119–124, 1983.

[56] M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, Los Altos, California, 1987.

[57] M.R. Genesereth. The use of design descriptions in automated diagnosis. In D.G. Bobrow, editor, *Qualitative reasoning about physical systems*, pages 411–436. MIT Press, Cambridge, Massachusetts, 1982.

[58] R. Goebel. A sketch of analogy as reasoning with equality hypotheses. In E. K. Jantke, editor, *Analogy and Induction, Lecture Notes in Computer Science.* Springer Verlag, Berlin, June 1989.

[59] R. Goebel, K. Furukawa, and D. Poole. Using definite clauses and integrity constraints as the basis for a theory formation approach to diagnostic reasoning. In *Proceedings of the Third International Conference on Logic Programming*, pages 211–222, London, England, July 14-18 1986. Imperial College.

[60] R. Goebel and S.D. Goodwin. Applying theory formation to the planning problem. In *Proceedings of the AAAI Workshop on The Frame Problem in Aritifical Intelligence*, pages 207–232, Lawrence, Kansas, April 12-15 1987.

[61] Randy Goebel. Complexity and implementability of non-monotonic reasoning systems. manuscript, 1989.

[62] S.D. Goodwin. *Statistically Motivated Defaults.* PhD thesis, Department of Computing Science, University of Alberta, Edmonton, Alberta, Fall, 1991.

[63] S.D. Goodwin and R. Goebel. Theory preference based on persistence. Research report CS-86-34, Department of Computer Science, University of Waterloo, Waterloo, Ontario, September 1986.

[64] S.D. Goodwin and R. Goebel. Non-monotonic reasoning in temporal domains: the knowledge independence problem. In M. Reinfrank, J. de Kleer, and M.L. Ginsberg, editors, *Non-Monotonic Reasoning*, number 346 in Lecture Notes in Artificial Intelligence, pages 187–201. Springer Verlag, Berlin, 1989.

[65] C. Green. Theorem-proving by resolution as a basis for question answering systems. In B. Meltzer aand D. Michie, editor, *Machine Intelligence*, volume 4, pages 183–205. Edinburgh University Press, 1969.

[66] S. Hanks and D.V. McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proceedings of the AAAI-86 Conference*, pages 328–333, Philadelphia, Pennsylvania, August 11-15 1986. University of Pennsylvania.

[67] R. M. Haralick and G. L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[68] D. Harel. *ALGORITHMICS The spirit of Computing.* Addison-Wesley Publishing Company, Menlo Park, 1987.

[69] W. S. Havens. Echidna constraint reasoning system: Programming specifications. In **Computational Intelligence III** — *International Conference on Computational Intelligence*, Milano, Italy, September 24-28 1990.

[70] P.J. Hayes. Computation and deduction. In *Procedings of the Symposium on the Mathematical Foundations of Computer Science*, pages 105–117, Czechoslovakian Academy of Sciences, 1973.

[71] P.J. Hayes. Some problems and non-problems in representation theory. In *Procedings of AISB Summer Conference*, pages 63–79, Sussex, England, 1974.

[72] P.J. Hayes. In defence of logic. In *Proceedings of the Fifth IJCAI*, pages 559–565, Cambridge, Massachusetts, August 22-25 1977. MIT.

[73] C.G. Hempel. *Aspects of scientific explanation and other essays in the philosophy of science*. The Free Press, New York, 1965.

[74] C. Hewitt. PLANNER: A language for proving theorems in robots. In *Proceedings of the IJCAI-71*, pages 295–301, London, England, 1971.

[75] C. Hewitt. Description and theoretical analysis (using schemata) of planner: a language for proving theorems and manipulating models in robot. Technical Report TR-258, Artificial Intelligence Laboratory, MIT, Cambridge, MA, 1972.

[76] D. Hilbert and W. Ackermann. *Principles of Mathematical Logic*. Chelesea Publishing Company, NewYork, 1950.

[77] K. Inoue. Procedural interpretation for an extended ATMS. Research Report ICOT TR-547, ICOT Research Center, Tokyo, Japan, 1990.

[78] M. Ishizuka and T. Matsuda. Knowledge acquisition mechanisms for a logical knowledge base including hypotheses. *Journal of Knowledge-based Systems*, 3(2):77–86, June 1990.

[79] D.J. Israel. What's wrong with non-monotonic logic? In *Proceedings of the AAAI-80 Conference*, pages 99–101, Stanford, California, August 18-21 1980. Stanford University.

[80] W.K. Jackson. A theory formation framework for learning by analogy. Master's thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, December 1986.

[81] J. Jaffer and J-L. Lassez. Constraint logic programming. In *Proceedings of 14th ACM-POPL Conference*, Munich, January 1987.

[82] D. Kaplan. What Is Russell's Theory of Description? In D. Davison and G. Harman, editors, *The Logic of Grammar*, pages 210–217. Dickenson Pub. Co. Inc., Encino, California, 1975.

[83] H. Kautz. The logic of persistence. In *Proceedings of the AAAI-86 Conference*, pages 401–405, Philadelphia, Pennsylvania, August 11-15 1986. University of Pennsylvania.

[84] A. Kean and G. Tsiknis. An incremental method for generating prime implicants/implicates. *Journal of Symbolic Computation*, Feb. 1990.

[85] C. Kellog, P. Klahr, and L. Travis. Deductive methods for large data bases. In *Proceedings of the Fifth IJCAI*, pages 203–209, Cambridge, Massachusetts, August 22-25 1977. MIT.

[86] K. Konolige. A deductive model of belief. In *Proceedings of the IJCAI-83*, pages 377–381, Karlsruhe, West Germany, August 1983.

[87] K. Konolige. Belief and incompleteness. In J. R. Hobbs and R. C. Moore, editors, *Formal Theories of the Commonsense World*. Ablex, Norwood, NJ, 1985.

[88] R. Kowalski. *Logic for Problem Solving*. North-Holland, New York, 1979.

[89] J.E. Laird, P.S. Rosenbloom, and A. Newell. Towards chunking as a general learning mechanism. In *Proceedings of AAAI-84*, Austin, TX, 1984.

[90] C. Lassez, K. McAloon, and R. Yap. Constraint logic programming and option trading. *IEEE Expert*, Fall issue 1987.

[91] H. J. Levesque. A logic of implicit and explicit belief. In *Proceedings of AAAI-84*, pages 198–202, Austin, TX, 1984.

[92] H. J. Levesque. Making believers out of computers. *Artificial Intelligence*, 30(1):81–108, 1986.

[93] H. J. Levesque and R. J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3(2):78–93, 1987.

[94] H.J. Levesque. Logic and the complexity of reasoning. Research Report KRR-TR-89-2, University of Toronto, Toronto, Canada, 1989.

[95] H. Levesque (ed.). Taking issue forum: A critique of pure reason. *Computational Intelligence*, 3(3):147–227, 1987.

[96] V. Lifschitz. Pointwise circumscription: Preliminary results. In *Proceedings of the AAAI-86 Conference*, pages 406–410, Philadelphia, Pennsylvania, August 11-15 1986. University of Pennsylvania.

[97] J. W. Lloyd. Directions for meta-programming. In *Proceedings of the International Conference on FGCS*, Tokyo, Japan, 1988.

[98] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, New York, 1984.

[99] J. W. Lloyd (ed.). Proceedings of the workshop on meta-programming in logic programming. June 1988.

[100] D.W. Loveland. *Automated theorem proving: a logical basis.* North-Holland, Amsterdam, The Netherlands, 1978.

[101] A. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[102] A. Mackworth. Constraint satisfaction. Technical Report 85-15, Department of Computer Science, University of British Columbia, Vancouer, B.C., September 1985.

[103] B. Magee. *Popper.* Fontana Press, London, England, 1973.

[104] D. Makinson. How to give it up: a survey of some formal aspects of the logic of theory change. *Synthese*, 62(4):347–363, 1985.

[105] Z. Manna and R. Waldinger. Special relations in automated deduction. *J. ACM*, 33(1):1–59, January 1986.

[106] J. P. Martins. The truth, the whole truth, and nothing but the truth: An indexed bibliography to the literature of truth maintenance systems. *AI Magzine*, (7):7–25, 1990.

[107] S. Matwin and T. Peitrzykowaski. Intelligent backtracking in plan-based deduction. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, PAMI-7(6):682–692, 1985.

[108] D. McAllester. A three valued truth maintenance system. AIM No. 473, Artificial Intelligence Laboratory, MIT, Cambridge, MA, 1978.

[109] D. McAllester. An outlook on truth maintenance. AIM No. 551, Artificial Intelligence Laboratory, MIT, Cambridge, MA, 1980.

[110] D. McAllester. Truth Maintenance. In *Proceedings of AAAI-90*, pages 1109–1116, Boston, USA, August 1990.

[111] J. McCarthy. The Advice Taker. In *Proceedings of Symposium on Mechaniszation of Thought Processes*, volume I, pages 77–84, London, 1958.

[112] J. McCarthy. Epistemological Problems of Artificial Intelligence. In *Proceedings of IJCAI-77*, pages 1038–1044, Cambridge, Massachusetts, 1977.

[113] J. McCarthy. Circumscription A form of non-monotonic reasoning. *Artificial Intelligence*, 13(1 & 2):27–39, 1980.

[114] J. McCarthy. Programs with Common Sense. In R. J. Brachman and H. J. Levesque, editors, *Readings in Knowledge Representation*, pages 299–307. Morgan Kaufmann, 1985. [Origonally appeared in *Semantic Information Processing* edited by M. Minsky, MIT press, 1968.].

[115] J. McCarthy. Applications of circumscription to formalising common-sense knowledge. *Artificial Intelligence*, 28(1):89–116, 1986.

[116] J. McCarthy and P. J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinberg University Press, 1969.

[117] D.V. McDermot. A critique of pure reason. *Computational Intelligence*, 3(3), 1987.

[118] D.V. McDermott. Tarskian semantics, or no notation without denotation. *Cognitive Science*, 2(3), 1978.

[119] D.V. McDermott and J. Doyle. Non-monotonic logic I. *Artificial Intelligence*, 13(1&2):41–72, 1980.

[120] M. Minsky. A framework for representing knowledge. In P. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, NewYork, 1975.

[121] M. Minsky and S. Papert. Progress report on artificial intelligence. Technical Report AI Memo 252, Artificial Intelligence Laboratory, MIT, Cambridge, MA, 1972.

[122] T. Mitchell. Generalisation as search. *Artificial Intelligence*, 18:203–226, 1982.

[123] U. Montanari. Networks of constraints: fundamental properties of and applications to picture processing. *Information Science*, pages 95–132, 1974.

[124] R.C. Moore. The role of logic in knowledge representation and commonsense reasoning. In R.J. Brachman and H.J. Levesque, editors, *Readings in Knowledge Representation*, pages 335–341. Morgan Kaufmann, Los Altos, California, 1985. [originally published in the proceedings of AAAI-82].

[125] R.C. Moore. Semantic considerations on nonmonotonic logic. *Artificial Intelligence*, 25, 1985.

[126] E. Neufeld and D. Poole. Combining logic and probability. *Computational Intelligence*, 4(1):98–99, February 1988.

[127] A. Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982.

[128] P. F. Patel-Schneider. A decidable first-order logic for knowledge representation. In *Proceedings of the IJCAI-85*, pages 455–458, Los Angels, CA, 1985.

[129] P. F. Patel-Schneider. A hybrid, decidable, logic-based knowledge representation system. *Computational Intelligence*, 3(2):64–77, 1987.

[130] G. Polya. *How to Solve It: A New Aspect of Mathetmatical Method.* Princeton University Press, Princeton, New Jersey, second, 1957 edition, 1954.

[131] D. Poole. Default reasoning and diagnosis as theory formation. Research report CS-86-08, Department of Computer Science, University of Waterloo, Waterloo, Ontario, March 1986.

[132] D. Poole. Fixed predicates in default reasoning. Research Report CS-87-11, Department of Computer Science, University of Waterloo, Waterloo, Ontario, February 1987.

[133] D. Poole. Explanation and prediction: an architecture for default and abductive reasoning. Technical report, Department of Computer Science, University of Waterloo, Waterloo, Ontario, February 1988.

[134] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36(1):27–47, 1988.

[135] D. Poole. Explanation and prediction: an architecture for default and abductive reasoning. *Computational Intelligence*, 5(2):97–110, May 1989.

[136] D. Poole, R. Goebel, and R. Aleliunas. Theorist: A logical reasoning system for defaults and diagnosis. In N.J. Cercone and G. McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, pages 331–352. Springer Verlag, New York, 1987.

[137] D.L. Poole. On the comparison of theories: Preferring the most specific explanation. In *IJCAI-85*, pages 144–147, Los Angeles, California, August 16-18 1985. UCLA.

[138] D.L. Poole. Defaults and conjectures: Hypothetical reasoning for explanation and prediction. Research Report CS-87-54, Department of Computer Science, University of Waterloo, Waterloo, Ontario, October 1987.

[139] D.L. Poole. Variables in hypotheses. In *Proceedings of the Tenth IJCAI*, pages 905–908, Milan, Italy, August 23-28 1987.

[140] H. E. Jr. Pople. Coming to grips with the multiple-diagnosis problem. In Kenneth F. Schaffner, editor, *The Logic of Discovery and Diagnosis in Medicine*, pages 181–198. University of California Press, Brekeley, 1985.

[141] K.R. Popper. *The Logic of Scientific Discovery.* Harper Torchbooks, New York, 1959.

[142] K.R. Popper. *Conjectures and Refutations: The Growth of Scientific Knowledge.* Routledge & Kegan Paul Limited, LONDON, 1963.

[143] W.V.O. Quine and J.S. Ullian. *The Web of Belief.* Random House, New York, 1978.

[144] R. Reiter. On closed-world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, 1978.

[145] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1&2):81–132, 1980.

[146] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[147] R. Reiter and J. de Kleer. Foundations of assumption-based truth maintenance systems (preliminary report). In *Proceedings of AAAI-87*, pages 183–187, Seattle, Washington, July 13-17 1987.

[148] R. Reiter and A. Mackworth. The logic of depiction. Technical Report 87-24, Department of Computer Science, University of British Columbia, Vancouer, B.C., September 1987.

[149] N. Rescher. *Hypothetical Reasoning.* North-Holland, Amsterdam, The Netherlands, 1964.

[150] N. Rescher. *Scientific Explanation.* Collier-MacMillan, Toronto, Ontario, 1970.

[151] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[152] A. Sattar. A hypothesis-based approach to rational belief revision. [in preparation], 1991.

[153] A. Sattar and R. Goebel. Using crucial literals to select better theories. Technical Report TR-89-27, Department of Computing Science, Edmonton, Alberta, 1989. [Revised version has appeared in *Computational Intelligence*, February, 1991].

[154] A. Sattar and R. Goebel. An incremental nonmonotonic theorem prover. In **Computational Intelligence III** — *International Conference on Computational Intelligence*, Milano, Italy, September 24-28 1990.

[155] A. Sattar and R. Goebel. On the efficiency of logic-based diagnosis. In *Proceedings of the Third International Conference on Industrial and Engineering Applications of AI/Expert Systems*, volume 1, pages 23–31, Charleston, South Carolina, USA, July 15-18 1990. ACM Press.

[156] A. Sattar and R. Goebel. A uniform framework for meta-reasoning, constraint solving and truth maintenance systems: A preliminary report. In *Proceedings of KBCS-90*, Pune, India, Dec. 13-15 1990. Narosa Publishing Co.

[157] A. Sattar and R. Goebel. Constraint Satisfaction in Hypothetical Reasoning. Technical Report TR-91-13, Department of Computing Science, Edmonton, Alberta, June 1991.

[158] A. Sattar and R. Goebel. Meta-reasoning: An incremental compilation approach. In *Proceedings of the Seventh International Conference on Data Engineering*, pages 140–149, Kobe, Japan, April 8-12 1991.

[159] A. Sattar and R. Goebel. Reducing explanation space for circumscriptive theorem provers. [in preparation], November 1991.

[160] A. Sattar and R. Goebel. Using crucial literals to select better theories. *Computational Intelligence*, 7(1):11–22, February 1991.

[161] A. Sattar, S. Goodwin, and R. Goebel. What is in consistency tree: A uniform treatment of theory selection. In *Proceedings of the Pacific RIM International Conference on Artificial Intelligence*, Nagoya, Japan, November 14-18 1990.

[162] L.K. Schubert, M.A. Papalaskaris, and J. Taugher. Determining type, part, color, and time relationships. *Computer*, 16(10):53–60, 1983.

[163] H. Seki and A. Takeuchi. An algorithm for finding a query which discriminates competing hypotheses. Techincal report TR-143, Institute for New Generation Computer Technology, Tokyo, Japan, October 1985.

[164] B. Selman. Computing exaplanations. In *Working Notes on Symposium: Automated Abduction*, pages 82–85, AAAI Spring Symposium Series, Stanford, March 1990. Extended Abstract.

[165] B. Selman and H. Kautz. The complexity of model-preference default theories. In *Proceedings of the CSCSI-88*, Edmonton, Alberta, 1988.

[166] M. Sergot. A query-the-user facility for logic programming. Research report DOC 82/18, Imperial College of Science and Technology, London, England, 1982.

[167] E.Y. Shapiro. *Algorithmic Program Debugging.* MIT Press, Cambridge, Massachusetts, 1982.

[168] Y. Shoham. Chronological ignorance: time, nonmonotonicity, necessity, and causal theories. In *Proceedings of the AAAI-86 Conference*, pages 389–393, Philadelphia, Pennsylvania, August 11-15 1986. University of Pennsylvania.

[169] B. C. Smith. *Reflections and Semantics in a Procedural Language*. PhD thesis, M.I.T., Cambridge, MA, 1982. [Technical Report MIT/LCS/TR-272].

[170] L. Sterling, A. Bundy, R. O'Keefe, and B. Silver. Solving symbolic equations with PRESS. In J. Calmet, editor, *Computer Algebra*, number No. 144 in Lecture Notes in Computer Science. Springer Verlag, 1982.

[171] M. E. Stickel. Theory resolution: building-in nonequational theories. In *Proceedings of AAAI-83*, pages 391–397, Washington, D.C., 1983.

[172] M. E. Stickel. Automated deduction by theory resolution. In *Proceedings of the IJCAI-85*, pages 148–154, Los Angels, CA, 1985.

[173] J. Stillman. The complexity of horn theories with normal unary defaults. In Peter F. Patel-Schneider, editor, *Proceedings of the Eighth Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, pages 92–97, Ottawa, Ontario, Canada, May 1990.

[174] G. J. Sussman and G. L. Steele. Constraint:a lanuage for expressing almost-heirarchical descriptions. *Artificial Intelligence*, 14:1–39, 1980.

[175] A. Takeuchi and K. Furukawa. Partial evaluation of prolog programs and its application to meta programing. Research Report ICOT TR-126, ICOT Research Center, Tokyo, Japan, 1989. [Original Japanese version in Logic Programming Conference'85].

[176] S-Å. Tärnlund. Horn clause computability. *BIT*, 17:215–226, 1977.

[177] A. Turing. On computable numbers with an application to the Entscheidungsproblem. In *Proceedings of London Math. Soc. Ser.*, volume 42, pages 230–265, 1936. [Correction appeared in Vol. 43, pp. 544-546].

[178] F. van Harmelen. A classification of meta-level architectures. In J. W. Lloyd, editor, *Proceedings of the workshop on Meta-Programming in Logic programming*, pages 81–92. University of Bristol, Bristol, June 1988.

[179] P. Van Hentenryck. A theoretical framework for consistency techniques in logic programming. In *Proceedings of the IJCAI-87*, pages 2–8, Milan, Italy, 1987.

[180] P. Van Hentenryck and M. Dincbas. Domains in logic programming. In *Proceedings of AAAI-86*, pages 759–765, Philadelphia, USA, August 1986.

[181] P. Van Hentenryck and M. Dincbas. Forward checking in logic programming. In *Proceedings of fourth international conference on Logic Programming*, Melbourne, Australia, May 1987.

[182] P. Voda. The constraint langauge trilogy: Semantics and computation. Technical report, Complete Logic Systems, North vancouver, BC, Canada, 1988.

[183] D. Waltz. Understanding line drawings of scenes with shadows. In P. H. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, New York, 1975.

[184] T. Winograd. Frame representations and the declarative/procedural controversy. In D.G. Bobrow and A. Collins, editors, *Representation and understanding*, pages 185–210. Academic Press, New York, 1975.

[185] B. Wong. Analogical reasoning based on hypothetical reasoning. Master's thesis, Department of Computing Science, University of Alberta, Edmonton, Alberta, November 1989.